Primi passi in Arduino con la scheda Genuino UNO

di Simone Majocchi

Entrate nel mondo del making e dell'interaction design con la piattaforma di sviluppo elettronica open-source più diffusa al mondo. È semplice, è intuitiva ed è alla portata di tutti. In una decina di ore sarete già in grado di progettare, scrivere da soli e caricare i vostri primi programmi sulla scheda Genuino UNO grazie ai semplici - ma efficaci - progetti ed esempi contenuti in questo libro.

BY NC ND

Testo Scritto da Simone Majocchi

La storia di questo testo:
Prima stesura e pubblicazione (Primi passi con Arduino UNO) **Dicembre 2011**
Ristampa aggiornata al **Giugno 2013**
Revisione e pubblicazione in CC (*Primi passi in Arduino con la scheda Genuino UNO*) **Gennaio 2016**

Impaginato dall'autore presso Aquarius Edizioni Srl - Viale Giustiniano 10 - 20129 Milano

Il logo e il marchio Genuino sono di proprietà di Arduino LLC, registrati negli Stati Uniti d'America e in altri stati.
Tutti gli altri marchi riportati appartengono ai legittimi proprietari; marchi di terzi, nomi di prodotti, nomi commerciali, nomi corporativi e società citati possono essere marchi di proprietà dei rispettivi titolari o marchi registrati d'altre società e sono stati utilizzati a puro scopo esplicativo ed a beneficio del possessore, senza alcun fine di violazione dei diritti di Copyright vigenti.

Le immagini riprodotte sono attribuite ai rispettivi autori dove possibile

Except where otherwise noted, this work is licensed under
http://creativecommons.org/licenses/by-nc-nd/3.0/

Primi passi in Arduino con la scheda Genuino UNO

di Simone Majocchi

CC BY NC ND

Nota sui componenti per gli esercizi proposti

Per svolgere le esercitazioni riportate in questo libro non avete bisogno di acquistare kit particolarmente costosi. I componenti che vi saranno necessario sono riassunti nella tabella sottostante e il loro costo, inclusa la scheda Genuino UNO non deve superare i 45 euro.

- **Scheda Genuino UNO (store.arduino.cc)**
- **Breadboard 400 punti**
- **3 LED rossi 5mm**
- **3 resistenze 330 ohm**
- **2 resistenze 180 ohm**
- **potenziometro rotatorio lineare 10k ohm**
- **slider lineare 10k ohm**
- **fotoresistenza 5mm 70-100k ohm**
- **1 resistenza 10k ohm**
- **trasduttore piezo passivo**
- **LED RGB catodo comune**
- **motore servo SG09**
- **spezzoni di filo per i collegamenti**

2011-2016 CC-BY-NC-ND

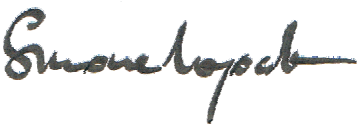
Dedicare dieci ore del proprio tempo all'apprendimento teorico e alla realizzazione di alcuni semplici progetti con la piattaforma di prototipazione elettronica Arduino e la scheda Genuino UNO è un investimento che permette di scoprire cosa si può creare con l'ingegno e l'inventiva. In questo tempo si potrà capire come un mondo apparentemente complesso e riservato agli esperti è stato invece semplificato e trasformato in un prodotto facile da usare - anche senza aver mai messo mano a un saldatore - e in grado di dare grandi soddisfazioni.
Rispetto ad altre soluzioni esistenti, la piattaforma Arduino conta su tre componenti uniche: un software per la programmazione, dell'hardware sotto forma di varie schede e una community di utenti. Il software si è evoluto negli anni grazie al contributo del Team di Arduino.cc che ha anche curato lo sviluppo delle varie schede, sempre rilasciate in open-source, da cui sono a loro volta nati migliaia di progetti e dispositivi. La community, infine, è il motore pensante di tutto questo: grazie ad essa il Team ha sempre trovato lo stimolo per migliorare, superando sempre i momenti di difficoltà e di duro lavoro che precedono il rilascio di un nuovo prodotto o la versione successiva dell'Arduino Software (IDE).

Il risultato è quindi sotto gli occhi di tutti: l'Arduino Software (IDE - Integrated Development Environment) è ad esempio un ambiente di sviluppo che è diventato nei suoi 10 anni di evoluzione uno standard per la programmazione ed è anche stato reso compatibile con schede e microcontrollori che non hanno a che fare direttamente con la piattaforma Arduino, ma ne condividono le funzionalità. Il circuito della scheda è stato reinterpretato, ridotto, ingrandito, integrato, migliorato e stravolto, rimanendo sempre compatibile con l'Arduino Software (IDE) e quindi mantenendo la sua natura open e programmabile. Designer, artisti, ricercatori e maker hanno trovato in questa piattaforma la soluzione più semplice ed efficace per realizzare dal singolo pezzo al prototipo, fino a una produzione industrializzata se necessario. Il tutto in poco tempo, una spesa ridotta e la soddisfazione di aver raggiunto l'obiettivo senza dover superare gli ostacoli di un hardware ostico e tutto da inventare.

Per entrare a far parte di questo mondo basta un po' di buona volontà, una mente aperta e la determinazione di chi non vuole più subire passivamente la tecnologia, ma vuole utilizzarla secondo le proprie esigenze. In questi anni ho potuto spiegare in corsi, seminari e workshop all'interno di scuole, fablab e centri di formazione quello che trovate in queste pagine. In una o due giornate ho avviato decine e decine di persone alla realizzazione di circuiti capaci di interagire con loro e con l'ambiente, trasformando stimoli in azioni. L'importante è superare i primi semplici ostacoli per trovare la confidenza necessaria a proseguire. Partiremo quindi con un po' di teoria e le basi indispensabili per capire come affrontare la programmazione, poi inizieremo a collegare qualche componente elettronico e ci avventureremo nel mondo degli algoritmi, con qualche spruzzata di elettronica di base. Una decina di capitoli che scorreranno via rapidamente e che vi porteranno a poter affrontare autonomamente qualche progetto tutto vostro. Nel 2016 potete contare su Google per scovare le informazioni su come risolvere questo o quel problema e ovviamente c'è tutto il patrimonio di esempi e tutorial che trovate ben documentati sul sito **Arduino.cc** che è gestito dal Team e dai co-fondatori di Arduino Massimo Banzi, David Cuartielles, Tom Igoe e David Mellis.

Buona lettura, ma soprattutto buon divertimento!

Simone Majocchi



2011-2016 CC-BY-NC-ND

Sommario

9

Perché Genuino Uno fa per te

Il nuovo "fai da te" tecnologico è basato sui microcontrollori - una storia che parte da lontano

15

Il poker tecnologico vincente

I tuoi assi sono: un personal computer, la scheda Genuino Uno, la tua creatività e dei componenti elettronici

21

Dalle idee al programma

Descrivete quello che volete fare, passo a passo, e arriverete al programma

35

Pensato per essere collegato a qualcosa

La scheda Genuino UNO nasce per essere collegata facilmente a tantissime cose, talmente tante che ad oggi nessuno le ha ancora individuate tutte

45

É ora di preparare la nostra cucina tecnologica

Abbiamo capito con chi abbiamo a che fare, ora prepariamo tutti gli elementi necessari a lavorare con Genuino UNO

2011-2016 CC-BY-NC-ND

La nostra prima realizzazione pratica: contiamo fino a tre

Se fossimo sui fornelli il primo piatto sarebbe un uovo all'occhio di bue, mentre in elettronica si comincia accendendo le luci. Noi accenderemo tre LED in sequenza

55

Basta poco per migliorare le cose

Aggiungiamo un potenziometro per variare la velocità dello scorrimento

65

Dopo le luci animate, ora creiamo dei suoni

Modifichiamo qualche elemento e cambiamo radicalmente il risultato: un generatore di suoni pilotato dalla luce ambientale

73

Il mondo è un arcobaleno di colori

Usiamo un LED RGB e il PWM per creare sedici milioni di colori in sequenza casuale

83

Verso una lampada per l'arcobaleno a comando

Usiamo un potenziometro per visualizzare tutti i colori dell'arcobaleno o per regolare l'intensità della luce bianca

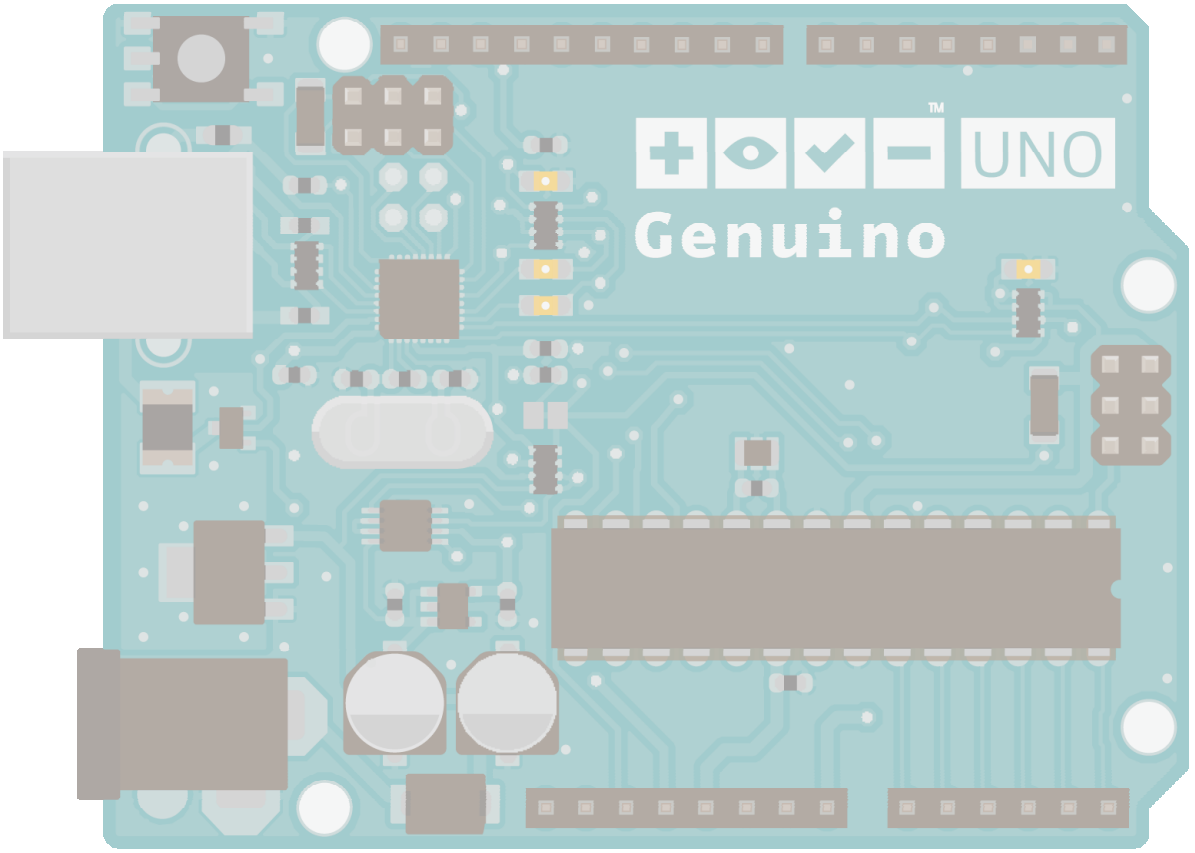
93

Servomotori per la robotica

Sempre con il PWM e un potenziometro possiamo realizzare un embrione di arto meccanico grazie a un servomotore

101

2011-2016 CC-BY-NC-ND



credit: Arduino.cc

2011-2016 CC-BY-NC-ND

Perché Genuino Uno fa per te

Il nuovo "fai da te" tecnologico è basato sui microcontrollori – una storia che parte da lontano

Oltre 35 anni fa erano in molto a dilettarsi con il saldatore, i circuiti stampati e i componenti discreti per creare amplificatori, luci psichedeliche, microspie e molti altri dispositivi elettronici. La cultura del "fai da te" elettronico era diffusa e con una robusta base commerciale. Negli anni '80 arrivarono gli home computer e l'attenzione migrò dall'elettronica alla programmazione, con tantissimi utenti che per passione scrivevano righe e righe di software per le applicazioni più disparate. Ai tempi, fra l'altro, il programmare era sia un hobby, sia un inizio di professione: i programmi disponibili erano ancora pochi e il "fai da te" in Basic era spesso l'unico modo per rispondere a un'esigenza specifica.

Programmare non è più un hobby

Parallelamente arrivarono i Personal Computer nelle aziende e anche in quel caso si doveva spesso far sviluppare il software per le proprie applicazioni.
In ogni caso, il computer e l'home computer erano i nuovi catalizzatori delle energie creative degli utenti più attivi e ben presto avere un computer non era più sinonimo di "programmo le mie applicazioni", ma cerco e utilizzo dei software scritti da altri. I più bravi scrivevano il software e i pigri ne usufruivano. Il "fai da te" diventava sempre più d'elite, mentre il "faccio anch'io" - con il software fatto da altri – era sempre più la norma.

Nel 1980 Sinclair Research Ltd, dopo una serie di prodotti e kit di successo, dalle calcolatrici agli orologi, presenta al mondo il suo home computer dotato di processore Z80 e interprete BASIC in 4K di ROM. La memoria per il codice era di 1K.

2011-2016 CC-BY-NC-ND

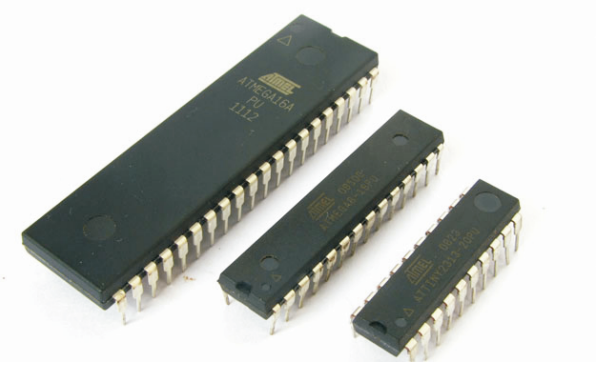
Il PC IBM, modello XT, ha determinato la nascita di molte realtà, da Microsoft che ha creduto nelle sue potenzialità, a tutte le aziende che hanno investito e sviluppato il business della produzione di software.

Programmare su un personal computer non era più facile come sugli home computer: interfacce grafiche raffinate, esigenze estetiche sempre più spinte e anche sistemi di sviluppo sempre più corposi.

Il PC cresce, ma il microprocessore torna piccolo

Fuori dai radar degli hobbysti stava, però, avanzando una nuova area tecnologica: quella dei microcontrollori destinati alla vita "solitaria" all'interno di qualche dispositivo; nascevano i cosiddetti "embedded" (dall'inglese "embed" che significa "integrare all'interno") per dare funzionalità evolute ai dispositivi elettrici ed elettronici. Con grande rapidità questi piccoli calcolatori autosufficienti hanno invaso le nicchie più disparate, fornendo di un tocco d'intelligenza una quantità enorme di dispositivi della vita quotidiana: dallo stereo di casa alla centralina della tuia automobile, dal forno a microonde al decoder Digitale Terrestre, dalla lavatrice alle lampade cambia colore a LED.

Aziende come Microchip, Atmel, ARM ed ST diventano i nuovi attori di questo millennio nel settore dell'elettronica, creando realtà industriali alternative e complementari ai colossi dei microprocessori tradizionali come AMD e INTEL.

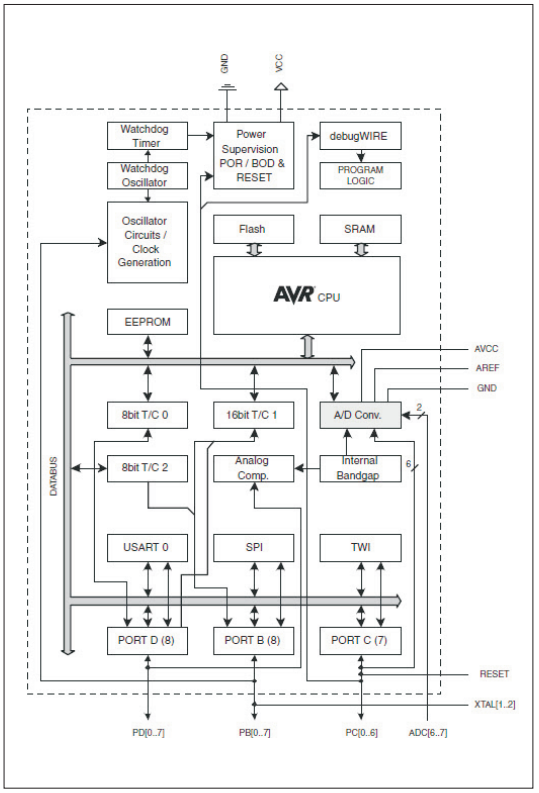


Oggi aziende come Microchip offrono intere famiglie di microcontrollori di ogni dimensione e potenza, con funzionalità specifiche (multimedia, sicurezza, comunicazione, ecc.) e generiche (ingressi e uscite analogici e digitali) tali da soddisfare una gamma molto vasta di applicazioni. Pochi dollari in più e un dispositivo può diventare "intelligente".

Torna il fai da te elettronico, ma intelligente e in kit

Queste caratteristiche e potenzialità non sono sfuggite agli hobbysti "duri e puri" del fai da te elettronico e già nei primi anni 2000 si è diffusa la cultura della programmazione di questi dispositivi per creare progetti più o meno sofisticati. In pratica, quello che una volta era un programma in Basic su un home computer, diventava un programma per un microcontrollore al quale con stagno, fili e compo-

2011-2016 CC-BY-NC-ND



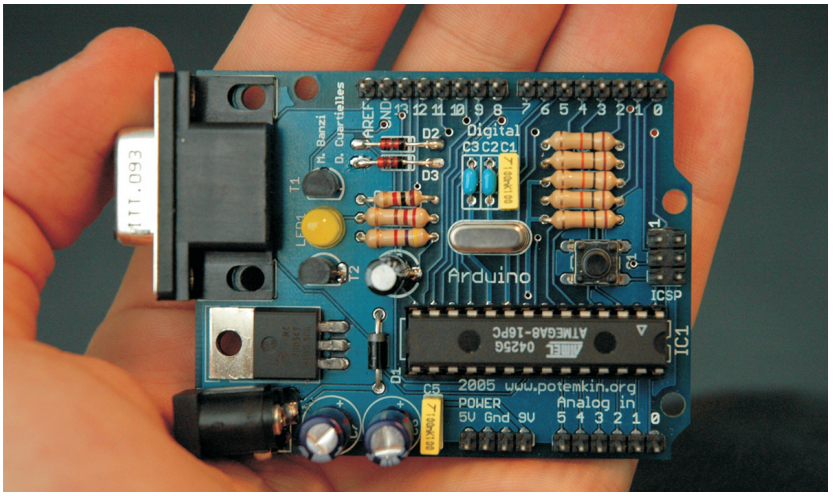
Il microcontrollore Atmega382 è alla base di molte schede Arduino e Genuino.

nenti si aggiungevano le parti necessarie al funzionamento. Sensori di luce, temperatura e movimento "intelligenti" per la domotica, giochi elettronici, centraline di controllo e molto altro prendevano vita grazie a schemi e programmi sviluppati da chi aveva già preso questo nuovo toro tecnologico per le corna, piegandolo alla dimensione del fai da te. Purtroppo, i chip richiedevano la programmazione su appositi dispositivi hardware, da interfacciare a un computer su cui era installato il sistema di sviluppo – spesso complicato quanto quelli per scrivere software su pc – riportando il tutto ad una elite. Il linguaggio di programmazione torna ad essere prima il "linguaggio macchina" – o assembler - con le istruzioni del processore da scrivere secondo sintassi e strutture per nulla amichevoli e intuitive. Per questi motivi, la creatività rimane limitata ai progetti sviluppati dalle riviste specializzate che ritrovano lettori proprio grazie ai microcontrollori e alle loro potenzialità.

"Deve diventare facile e alla portata di chi non è un esperto"

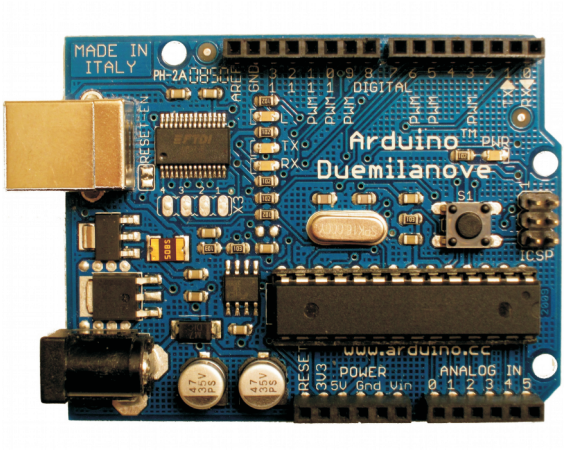
I numeri crescono sia a livello industriale, sia a livello hobbystico e l'interesse verso questi microcontrollori si sviluppa negli ambiti più diversi, inclusi quelli della ricerca e delle università. A Ivrea, un gruppo di ricercatori con la vocazione da "tinkerers" (in italiano "smanettoni" nell'accezione più positiva del termine) dedica creatività, visione e capacità alla creazione di una "soluzione" per sviluppare e lavorare facilmente con un microcontrollore, cercando di eliminare tutte le criticità che avevano impedito a questi dispositivi di entrare nel mondo del fai da te "povero" di conoscenze specifiche e tecnologiche, ma "ricco" di inventiva e creatività.

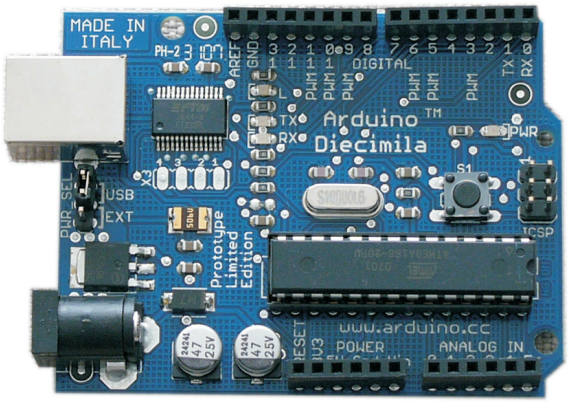
Nasce l'idea di Arduino: un microcontrollore con tutti i vari collegamenti esplicitati su una basetta economica e in open-source, da interfacciare a un computer (mac, windows, linux) per la programmazione tramite un normalissimo cavetto USB e soprattutto program-

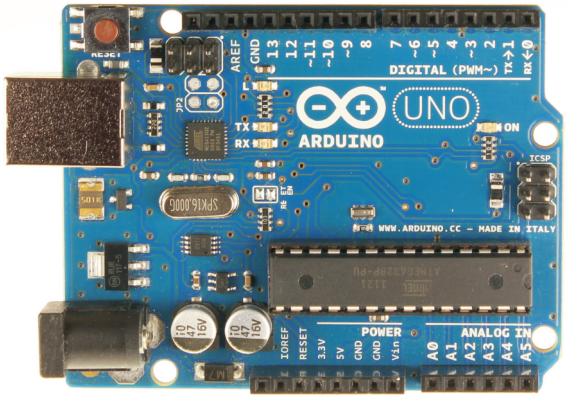


Una delle prime versioni di scheda Arduino, realizzata nel 2005. Foto by Nicholas Zambetti

2011-2016 CC-BY-NC-ND







Le schede che hanno sancito il successo della piattaforma di prototipazione elettronica Arduino.

mabile in un linguaggio adatto anche ai principianti grazie all'ambiente di sviluppo integrato scritto in Java. Arduino ambiva a diventare uno strumento alla portata di chiunque per la realizzazione di prototipi e progetti su piccola scala in cui la flessibilità e la potenza di un microcontrollore permettevano di raggiungere l'obiettivo senza spendere una fortuna.

Duemilanove, Diecimila e... "UNO"

A sei anni di distanza, il team che ha ideato Arduino può affermare di aver raggiunto l'obiettivo grazie alle scelte dell'open-source e dei Creative Commons (trovate la spiegazione approfondita di queste "filosofie tecnologiche" anche sul sito di Arduino stesso – arduino.cc) che hanno attirato l'attenzione di un significativo numero di utenti, sviluppatori, hobbisti, artisti e gente comune. Senza segreti e brevetti, il Team di Arduino ha permesso a numerose aziende di scommettere su questa piattaforma producendo le schede stesse, espansioni (gli shield) e altre periferiche dedicate.
Si è perso – apparentemente - un po' dello spirito del tinkerer che prende un oggetto comune e lo modifica aggiungendo dei fili e un Arduino, come un moderno McGyver, ma si guadagna in maturità della soluzione, che abbandona la precarietà della versione "beta" e approda a un design di hardware e software stabili – e per questo chiamato UNO. Dal primo rilascio la versione viene revisionata con iterazioni successive fino ad arrivare alla nota "Rev.3". Con questa versione, la scheda e il suo ambiente di sviluppo, sempre parte integrante della piattaforma, hanno un successo planetario e di fatto Arduino UNO diventa uno degli strumenti principali del maker.

Saldatore, stagno, componenti e scheda Genuino UNO: il nuovo trend "cool"

Per usare la piattaforma Arduino basta la scheda assieme a un personal computer e quindi chiunque può essere indipendente: il progetto non dipende dalla voglia di qualcun altro per realizzare una scheda dedicata e scrivere del codice assembler che viene venduto preprogrammato su un microcontrollore, che va acquistato e

2011-2016 CC-BY-NC-ND

Le schede ufficiali del Team Arduino.cc

Negli anni sono state prodotte una serie di schede con dimensioni e microcontrollori diversi; alcune sono ancora disponibili, altre sono ormai fuori produzione. Attualmente le schede che riportano il marchio **Arduino** sono prodotte per il solo mercato degli USA, mentre nel resto del mondo il marchio utilizzato è **Genuino**. Ecco l'assortimento attuale con le caratteristiche più importanti.

Fonte: sito **arduino.cc**

Nome	Processore	Tensione di input e alimentazione	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [KB]	SRAM [KB]	Flash [KB]	USB	UART
ArduinoBT	ATmega328P	5 V / 2.5-12 V	16 MHz	6/0	14/6	1	2	32	-	1
Due	ATSAM3X8E	3.3 V / 7-12 V	84 MHz	12/2	54/12	-	96	512	2 Micro	4
Esplora	ATmega32U4	5 V / 7-12 V	16 MHz	-	-	1	2.5	32	Micro	-
Ethernet	ATmega328P	5 V / 7-12 V	16 MHz	6/0	14/4	1	2	32	Regular	-
Fio	ATmega328P	3.3 V / 3.7-7 V	8 MHz	8/0	14/6	1	2	32	Mini	1
Gemma	ATtiny85	3.3 V / 4-16 V	8 MHz	1/0	3/2	0.5	0.5	8	Micro	0
Leonardo	ATmega32U4	5 V / 7-12 V	16 MHz	12/0	20/7	1	2.5	32	Micro	1
LilyPad	ATmega168V ATmega328P	2.7-5.5 V / 2.7-5.5 V	8MHz	6/0	14/6	0.512	1	16	-	-
LilyPad SimpleSnap	ATmega328P	2.7-5.5 V / 2.7-5.5 V	8 MHz	4/0	9/4	1	2	32	-	-
LilyPad USB	ATmega32U4	3.3 V / 3.8-5 V	8 MHz	4/0	9/4	1	2.5	32	Micro	-
Mega 2560	ATmega2560	5 V / 7-12 V	16 MHz	16/0	54/15	4	8	256	Regular	4
Mega ADK	ATmega2560	5 V / 7-12 V	16 MHz	16/0	54/15	4	8	256	Regular	4
Micro	ATmega32U4	5 V / 7-12 V	16 MHz	12/0	20/7	1	2.5	32	Micro	1
Mini	ATmega328P	5 V / 7-9 V	16 MHz	8/0	14/6	1	2	32	-	-
Nano	ATmega168 ATmega328P	5 V / 7-9 V	16 MHz	8/0	14/6	0.512 1	1 2	16 32	Mini	1
Pro	ATmega168 ATmega328P	3.3 V / 3.35-12 V 5 V / 5-12 V	8 MHz 16 MHz	6/0	14/6	0.512 1	1 2	16 32	-	1
Pro Mini	ATmega328P	3.3 V / 3.35-12 V 5 V / 5-12 V	8 MHz 16 MHz	6/0	14/6	0.512	1	16	-	1
Uno	ATmega328P	5 V / 7-12 V	16 MHz	6/0	14/6	1	2	32	Regular	1
Yùn	ATmega32U4 AR9331 Linux	5 V	16 MHz 400MHz	12/0	20/7	1	2.5 16MB	32 64MB	Micro	1
Zero	ATSAMD21G 18	3.3 V / 7-12 V	48 MHz	6/1	14/10	-	32	256	2 Micro	2
101	Intel® Curie	3.3 V/ 7-12V	32MHz	6/0	14/4	-	24kB	196kB	Regular	-
MKR1000	SAMD21 Cortex-M0+	3.3 V / 5 V	48 MHz	7/1	8/4	-	32kB	256kB	Micro	1

2011-2016 CC-BY-NC-ND

Il team

Il gruppo stabile di sviluppatori e progettisti del team fa riferimento al sito **https://www.arduino.cc** e i co-fondatori di Arduino che stanno proseguendo nello sviluppo della piattaforma assieme al team sono Massimo Banzi, David Cuartielles, Tom Igoe e David Mellis.

usato così com'è. Ciascuno diventa autore di soluzioni e grazie allo spirito di condivisione della community queste soluzioni sono messe a disposizione di chiunque tramite internet. Negli anni con la piattaforma open-source Arduino sono stati realizzati progetti di ogni genere ed è grazie ad essa che abbiamo le stampanti 3D a basso costo o i quardicotteri droni e mille altre innovazioni tecnologiche.

Dieci ore spese bene

Come resistere a questo rinascimento del fai da te tecnologico? Una volta c'era l'hobbysta elettronico, mentre oggi ci sono i maker che hanno la marcia in più dei microcontrollori. Non c'è progetto che grazie alla piattaforma Arduino non possa diventare possibile o acquistare una funzione in più. L'importante è impadronirsi dei concetti di base e superare quei piccoli scogli iniziali che anche Arduino – pur nella sua semplicità – vi porterà ad affrontare in queste prime dieci ore. Una volta che avrete superato questa fase, sarete voi a voler andare avanti.
Abbiamo preparato l'aperitivo e speriamo di farvi venire una gran fame di informazioni. Il Web potrà offrire molti piatti gustosi, anche se problemi linguistici e gusti diversi, assieme all'intrinseca complessità/confusione della Rete potrebbero disorientarvi. Non vi demoralizzate e partecipate alle community, ricordando che tutti gli "espertoni" che incontrerete hanno passato un periodo da neofiti come voi e per questo sapranno capire i vostri limiti e apprezzare la vostra iniziativa. Ricordate solo che sui forum, prima di chiedere, è buona educazione cercare e leggere quanto è stato già detto e scritto.
Google in questo sarà il primo strumento da utilizzare, cercando di inserire come termini di ricerca un po' di parole, così da ottenere risultati ragionevolmente pertinenti.
Ad oggi, su Google, le pagine che contengono la parola "Arduino" hanno superato i 36 milioni, come sono milioni gli utenti che scaricano l'Arduino Software (IDE) per programmare una scheda compatibile con la piattaforma.

Un po' di chiarezza sul nome

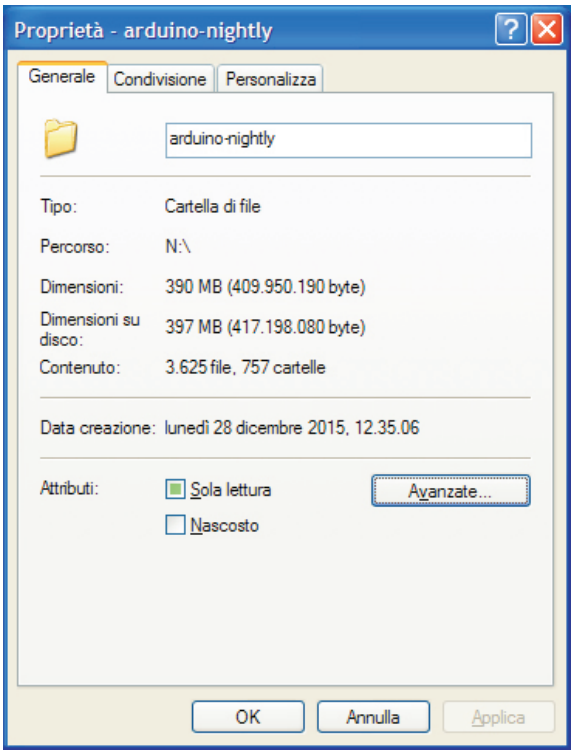
In attesa della soluzione circa la contesa del marchio "Arduino" - di proprietà di Arduino LLC negli Stati Uniti d'America – per i prodotti venduti al di fuori degli USA è stato creato il marchio "Genuino" da parte dei co-fondatori di Arduino M Massimo Banzi, David Cuartielles, Tom Igoe e David Mellis assieme al team Arduino.cc e alla community. Il marchio Genuino certifica l'autenticità delle schede e dei prodotti in linea con ta filosofia di open-hardware e open-source che ha sempre caratterizzato i valori fondanti di Arduino.
Genuino rende il progetto open-source Arduino disponibile a livello globale attraverso alleanze con I principali produttori locali in Asia, Europa, Sud America, Canada e Africa.

2011-2016 CC-BY-NC-ND

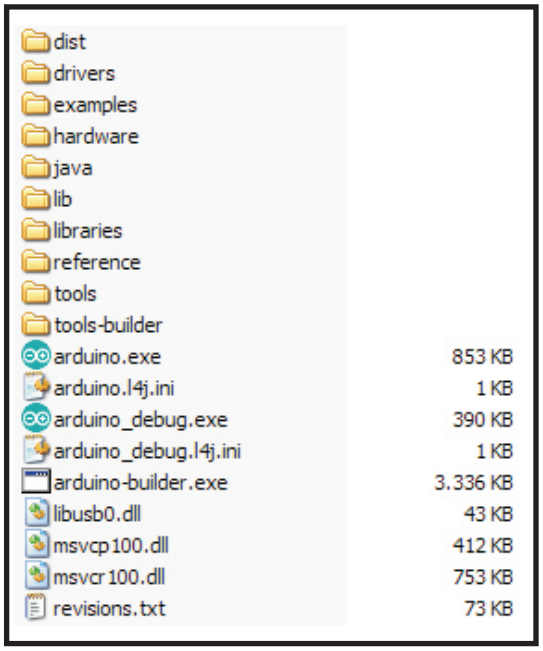
Il poker tecnologico vincente

I tuoi assi sono: un personal computer, la scheda Genuino Uno, la tua creatività e dei componenti elettronici

La soluzione realizzata per programmare la scheda Genuino UNO nasconde all'utente tutta una serie di passaggi, di configurazioni e di programmi che altrimenti avrebbero richiesto capacità e preparazione per essere messi a punto e utilizzati. Vi accorgerete di questo quando installerete l'IDE scaricabile gratuitamente dal sito Arduino.cc: oltre 3.500 file, molte directory e tutti gli eseguibili richiesti per svolgere le numerose operazioni che portano da uno "sketch" al codice eseguibile su un microcontrollore con core AVR o un altro core supportato.
Il grande lavoro svolto dal Team di Arduino.cc è stato quindi in gran parte creare questo ambiente di sviluppo (Integrated Development Environment – IDE). Ad oggi siamo giunti alla release 1.6.7 grazie alla quale, solo apparentemente, sembra tutto molto semplice e lineare, mentre dietro all'interfaccia scritta in Java si trovano quelle migliaia di file che permettono al tuo computer di trasformare il tuo programma nel modo opportuno e riducendo al minimo l'insorgere di problemi di compilazione (il processo che trasforma il programma scritto con il linguaggio ad alto livello in codice macchina adatto al microcontrollore presente sulla scheda).



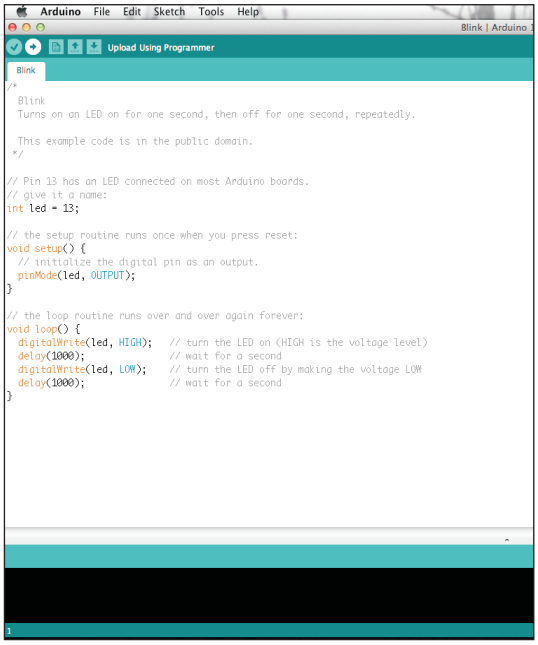
Oltre alla versione standard di Arduino Software (IDE), esiste anche quella aggiornata ogni ora, dedicata a chi desidera avere l'ultima release disponibile. Questa versione recepisce tutte le modifiche gestite tramite Github interagendo con la community; anticipa le build stabili che vengono rilasciate periodicamente.

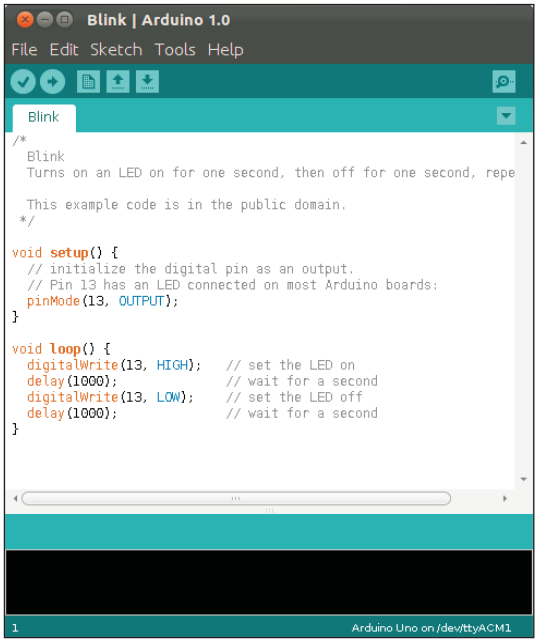


Oltre a creare il codice eseguibile e a scaricarlo sul microcontrollore della scheda, l'IDE permette anche di raccogliere informazioni di ritorno dalla scheda stessa. Il computer può quindi utilizzare Genuino UNO come periferica dalla quale acquisisce informazioni sotto forma di testo, inviato sulla porta seriale virtuale che il driver della scheda installa in modo automatico. Grazie a questo, l'interattività e i "sensi" che possiamo dare a Genuino UNO possono essere trasferiti al computer con grande semplicità.

Il vostro computer, sia esso un pc con Windows, un Macintosh o un sistema

2011-2016 CC-BY-NC-ND





Il look e le funzionalità di Arduino Software (IDE) sono identici su tutte le piattaforme supportate. Dall'alto Windows, OSx e Linux.

con Linux, è un elemento indispensabile per poter creare con l'Arduino Software (IDE).
Una volta ultimato il processo creativo e di programmazione, Genuino UNO potrà anche operare autonomamente eseguendo il software e gestendo l'hardware collegato senza più alcun collegamento al pc.

La scheda Genuino UNO

Il suo cuore è un microcontrollore della famiglia AVR di Atmel, ma la sua struttura - affinata nel corso di oltre cinque anni - la rende ad oggi la soluzione più economica e flessibile per fare qualcosa di interattivo, elettronico e moderno.
Con Genuino UNO non c'è bisogno di un programmatore dedicato per microcontrollori in quanto la scrittura del codice nella memoria flash avviene tramite il collegamento USB al PC. Anche l'alimentazione può essere direttamente fornita dal computer sempre tramite USB, mentre se Genuino Uno deve funzionare da solo, c'è un connettore per collegare un alimentatore esterno da 7 a 12V che viene poi regolato ai canonici 5V da un regolatore presente sulla scheda.
Con la sua schiera di porte e di pin di ingresso e uscita, Genuino UNO è in grado di acquisire informazioni da una miriade di sensori e gadget elettronici, potendo allo stesso tempo accendere luci e dispositivi, pilotare motori o emettere suoni.
Se funziona a corrente o è elettronico, in qualche modo potrà essere collegato a Genuino UNO, mentre se è meccanico, con un po' di fantasia e di meccanica applicata, sarà possibile metterlo in movimento.
Partendo da questi presupposti e da questa apertura possiamo aggiungere l'ingrediente successivo…

La vostra creatività

I programmi scritti con l'Arduino Software (IDE) non a caso si chiamano "sketch": in italiano sono i "bozzetti", gli schizzi che gli artisti fanno per mettere su carta con un dettaglio via via crescente le proprie idee e questo paragone si adatta bene a come si programma. Senza idee non ci sono neanche i bozzetti: il foglio resta bianco.

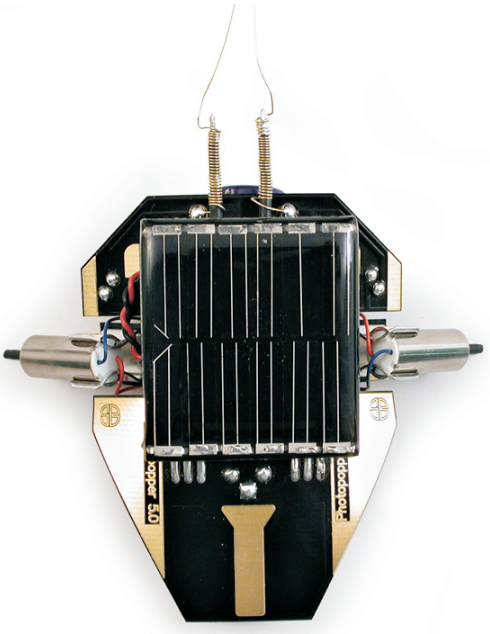
2011-2016 CC-BY-NC-ND

Come si fa in natura?

credit: Insect Lab

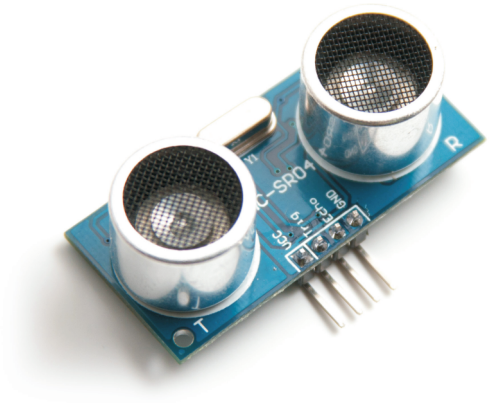
Dopo tutta questa tecnologia, vogliamo attirare la vostra attenzione sul lavoro fatto sin qui da madre natura per consentire a tutti gli esseri viventi di relazionarsi con il mondo che li circonda.
Un po' di ricerca sulla struttura degli organi di percezione degli animali, anche a livello di gestione degli stimoli per reagire, può essere illuminante al fine di sviluppare delle soluzioni efficaci e funzionali.
Molti animali possono contare su pochissima intelligenza, per non dire solo qualche neurone, eppure sono capaci di comportamenti che raggiungono lo scopo della sopravvivenza e della riproduzione.

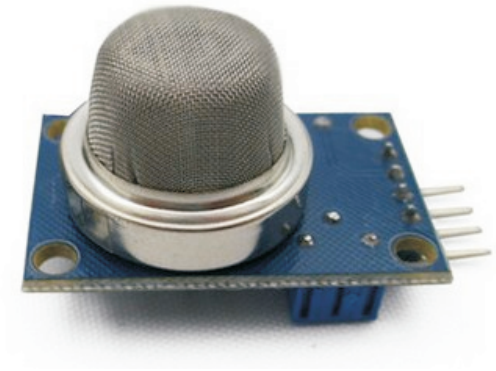
Anche il vostro Genuino UNO resta un mucchietto di elettronica se non aggiungete le vostre idee e la vostra creatività. Potete immaginarlo come uno strumento musicale sul quale potete inizialmente fare dei semplici giri di accordi per prendere confidenza; presto sentirete l'esigenza di interpretare qualche canzone fra quelle che preferite aggiungendo espressività all'esecuzione e infine vi sentirete pronti per creare qualcosa di originale.
Traducendo questo esempio in situazioni pratiche, vi aspettano i primi passi con quei semplici progetti che abbiamo selezionato per voi. Nulla di trascendentale, anzi, ma alla portata di tutti, anche dei veri neofiti.
Come accade per la musica, anche i programmi e i progetti con Genuino seguono dei filoni: luci e colori, Suoni e musica, sensori e monitoraggio dell'ambiente, aggiunta di funzioni a dispositivi esistenti, robotica e domotica, comunicazione e telecontrollo, per citare i più comuni. È assolutamente normale che ciascuno sia più o meno attratto da ciascuna di queste tematiche, come ciascuno preferisce alcuni generi musicali e non sopporta altri. È altrettanto indiscutibile che non si possa dire se qualcosa piace o non piace prima di averne fatto la conoscenza.
La vostra creatività e le vostre capacità di ideare progetti per Genuino UNO vanno quindi stimolate sperimentando un po' in tutti i vari campi applicativi. Leggete quello che altri hanno già fatto e provate a realizzare in pratica i progetti che più vi incuriosiscono o vi attirano.



Questo Photovore è in grado di spostarsi verso la fonte luminosa presente nell'ambiente, evitando anche gli ostacoli, eppure utilizza solo due semplici "neuroni" analogici stimolati da due occhi sensibili alla luce.
www.solarbotics.com

2011-2016 CC-BY-NC-ND





Sopra: il sensore di distanza a ultrasuoni HC-SR04; sotto, il sensore di gas infiammabili MQ-2.

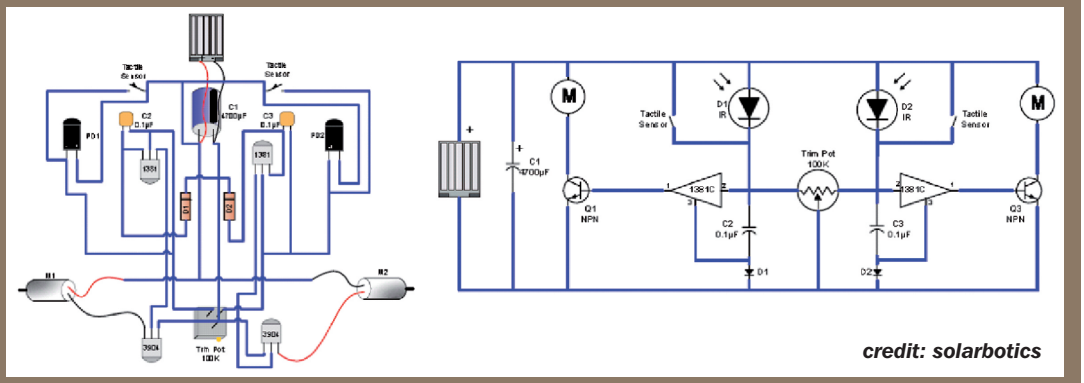
Quando avrete trovato i "generi" per i quali siete portati, date libero sfogo alla vostra creatività e iniziate a immaginare le vostre personalissime soluzioni, senza dimenticare che è originale anche la combinazione di "pezzi" creati da altri.

Quanti sensi ha Genuino UNO?

Collegando gli opportuni sensori potremmo dire che a parte il gusto, Arduino UNO dispone di tatto, udito, olfatto (limitato a certe sostanze) e vista; a questi si aggiungono poi dei "superpoteri" come l'orientamento, il calcolo delle distanze, la vista agli infrarossi e altro ancora.
Le tecnologie dei sensori avanzano ogni anno verso nuovi traguardi e sia sul fronte dei costi, sia su quello delle funzionalità, si stanno facendo passi da gigante. Se qualche anno fa un giroscopio e un accelerometro erano sensori riservati alle applicazioni professionali, ora con meno di una decina di euro è possibile avere lo stato dell'arte.
La scelta del tipo e del numero di sensori dipende evidentemente dall'applicazione e dal buon senso: utilizzare più sen-

Due "neuroni elettronici" per inseguire la luce

Ecco come funziona il Fotovoro di Solarbotics, dove la differenza di illuminazione fra i due sensori determina quale motore deve essere azionato e porta quindi ad un movimento che, passo dopo passo, fa avvicinare il piccolo robot alla sorgente luminosa. Con Genuino UNO sareste capaci di fare lo stesso?



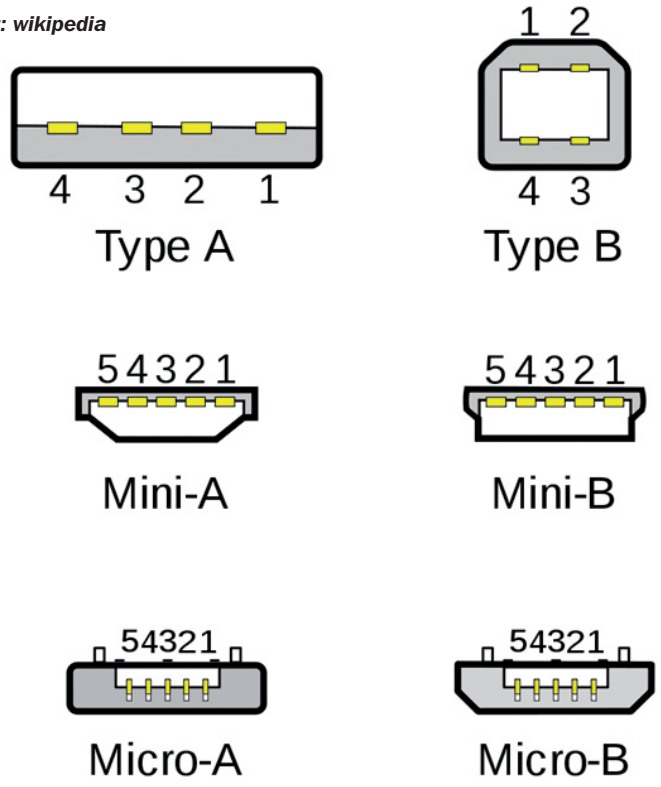
2011-2016 CC-BY-NC-ND

sori per qualcosa di omnidirezionale o ambientale (per esempio più accelerometri su un unico sistema solidale, oppure più sensore di vapori tossici per un unico ambiente) non è una grande idea, a meno di specifiche esigenze, mentre altri sensori possono migliorare le prestazioni con l'aumento del loro numero.
Va ricordato che Genuino UNO, pur con la sua notevole flessibilità, ha dei limiti precisi in termini di quante porte sono disponibili per gli ingressi e le uscite, assieme alle limitazioni del loro tipo. Per questo motivo, se guardate la tabella dei vari tipi di scheda, troverete l'indicazione del numero di porte disponibili e se il vostro progetto richiede molti collegamenti, è il caso di valutare le versioni opportune, oppure potete intraprendere la strada a più schede Genuino UNO che colloquiano fra loro e si dividono i compiti.
Un esempio di come l'architettura di Genuino sia adeguata ad applicazioni complesse è quello delle attuali stampanti 3D: il loro cuore, nella maggior parte dei casi, è una Genuino MEGA2560 su cui è montato uno shield che contiene tutta la componentistica necessaria a pilotare i motori stepper, a gestire l'elemento riscaldante dell'estrusore e a leggere i vari sensori di temperatura e di fine corsa. Sempre la scheda è in grado di gestire uno schermo a cristalli liquidi, anche grafico, e leggere le schede SD con i file da stampare, rendendo così la stampante indipendente dal collegamento con un personal computer.

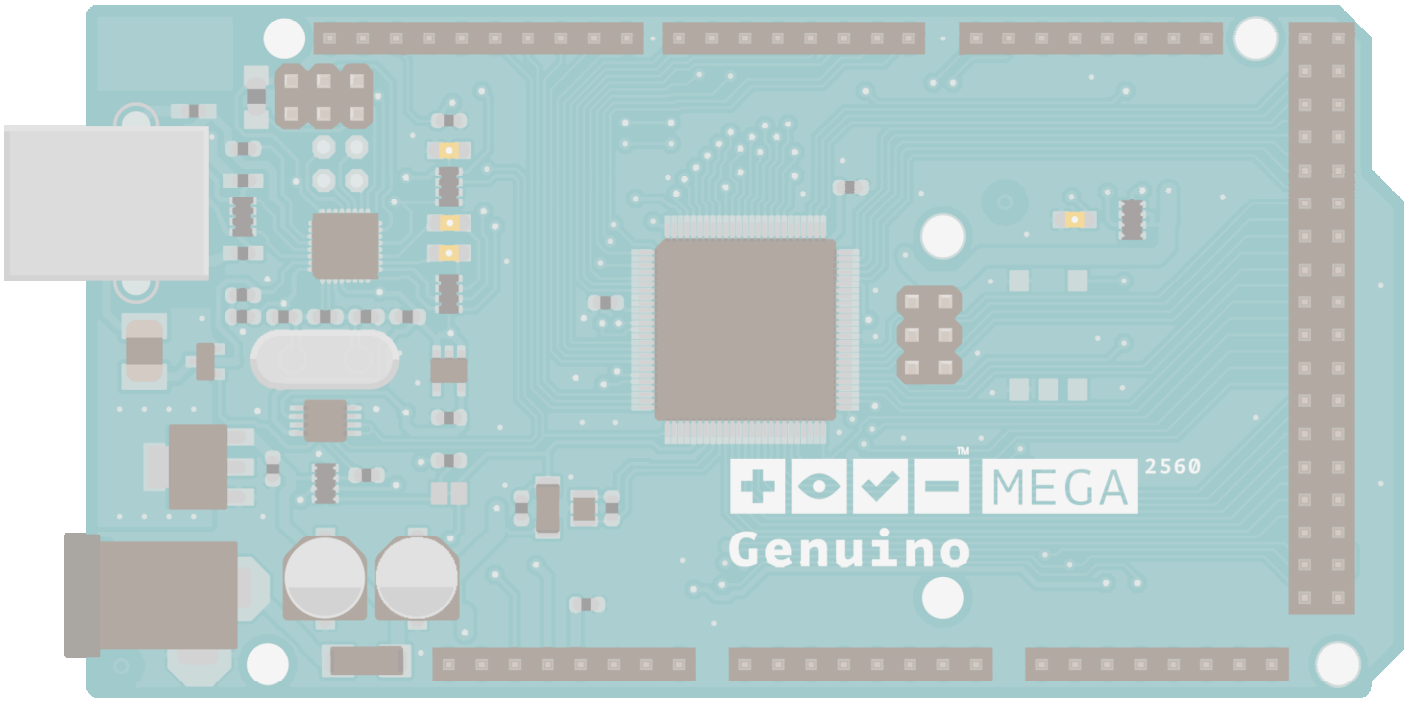
La porta USB

Da diversi anni le due porte classiche dei computer, quella parallela per le stampanti e quella seriale per i modem, sono state definite come obsolete, soprattutto sui computer portatili. Tutto è stato convertito allo standard Universal Serial Bus – USB grazie al quale collegare e utilizzare periferiche e dispositivi è diventato semplice e affidabile.
Oggi siamo arrivati alla terza versione di USB, retro compatibile con la versione 2.0 e 1.0, ma decisamente più veloce nel trasferimento dei dati.
Lo standard USB non definisce solo la forma e la parte elettrica della porta fisica di collegamento, ma dettaglia e definisce tutte le varie modalità di dialogo fra computer e periferica, protocollo e operatività.
Purtroppo negli anni sono state fatte varie versioni di connettore, cercando di ridurlo sempre di più. Per questo ogni tanto si può incontrare qualche problema di collegamento fra una porta del computer e un dispositivo, specie se questo è di recente produzione e i cavi a nostra disposizione sono un po' datati.
Il cavo da utilizzare con Genuino UNO è di tipo AB, mentre su altre schede della famiglia Genuino possiamo trovare anche i connettori di tipo Mini e Micro.

credit: wikipedia



2011-2016 CC-BY-NC-ND



credit: Arduino.cc

2011-2016 CC-BY-NC-ND

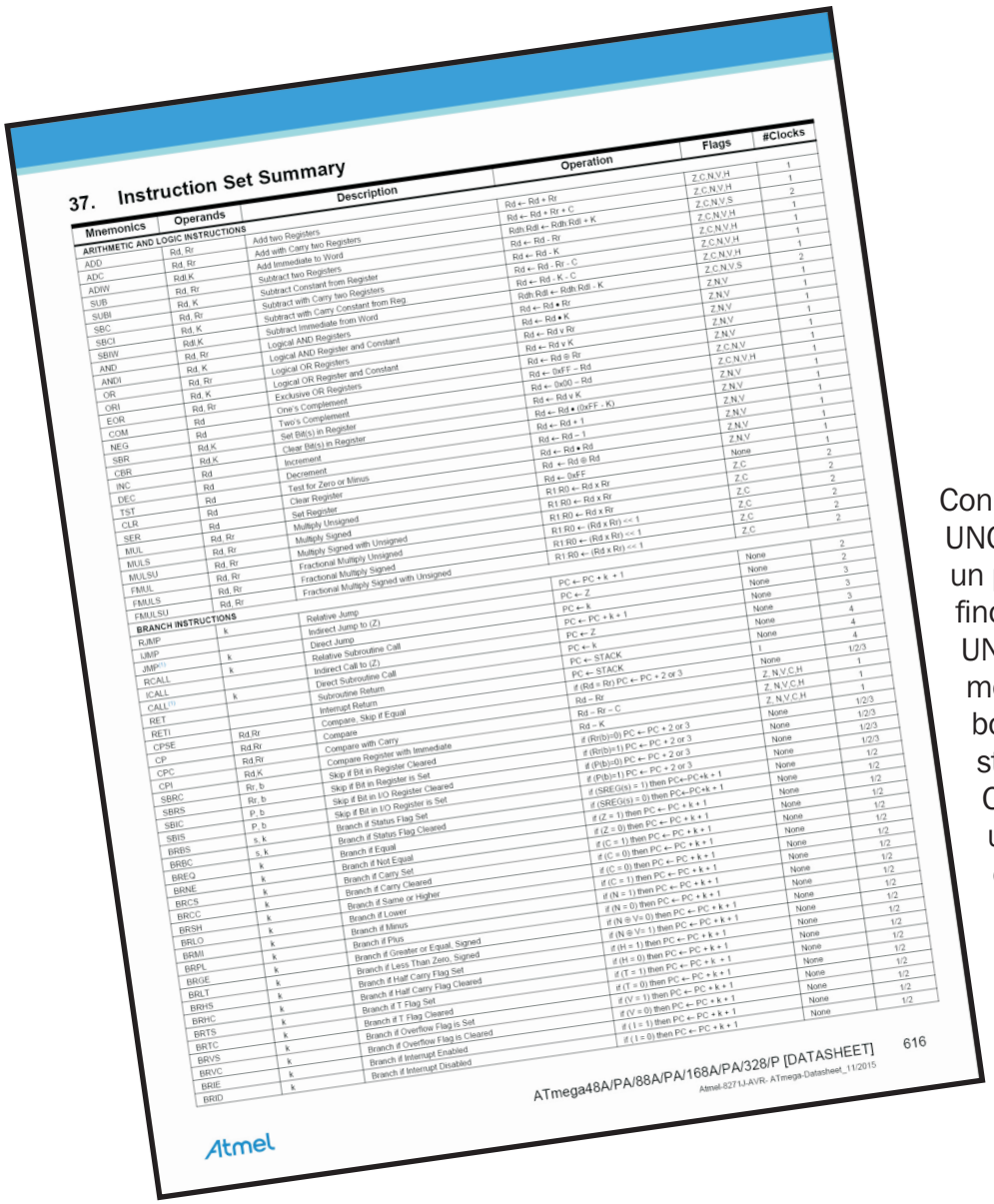
Dalle idee al programma

Descrivete quello che volete fare, passo a passo, e arriverete al programma

In elettronica si disegnavano gli schemi a blocchi per descrivere interamente le funzionalità del circuito e poi si popolava il tutto con i componenti; con i microcontrollori si descrive la serie di operazioni che vanno eseguite – anche in base agli eventi e alle rilevazioni dei sensori – fino a creare il flusso logico completo.
Questo livello di astrazione è indispensabile – e anche molto comodo – per dare l'ossatura al programma che scriverai nel linguaggio e con le modalità previste da Genuino UNO. Saltare questa fase per buttarsi a capofitto nella scrittura del codice potrà dare dei risultati nel breve termine, ma vi impedirà di mantenere tutto sotto controllo nei progetti di complessità crescente.
Usate un metodo per lavorare e cercate di rispettare anche l'esecuzione delle parti apparentemente più noiose: tutto questo è indispensabile – a nostro avviso – per poter crescere e superare le difficoltà che la programmazione degli algoritmi e dei progetti più sofisticati vi metterà di fronte.
Chi scrive questo libro ritiene che programmare sia ancora una forma d'arte, dove la creatività può fare la differenza. Le strade per arrivare alla soluzione sono tante e solo alcune sono particolarmente efficaci, mentre altre sono mediocri o poco efficienti.
Con i personal computer la programmazione del software è diventata ipertrofica, con applicazioni da migliaia e migliaia di righe di codice, magari destinate in gran parte ad abbellire una interfaccia grafica senza apportare nulla all'efficienza del programma stesso. D'altronde con centinaia di megabyte a disposizione, la compattezza del codice è qualcosa che è stato perso per strada da molti programmatori.

I linguaggi di programmazione disponibili oggi sono numerosi e la maggior parte è basata sugli oggetti.

2011-2016 CC-BY-NC-ND



37. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF − Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 − Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd − 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	None	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP[1]	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL[1]	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	None	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd − Rr	Z,N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd − Rr − C	Z,N,V,C,H	1
CPI	Rd, K	Compare Register with Immediate	Rd − K	Z,N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2

ATmega48A/PA/88A/PA/168A/PA/328/P [DATASHEET] 616

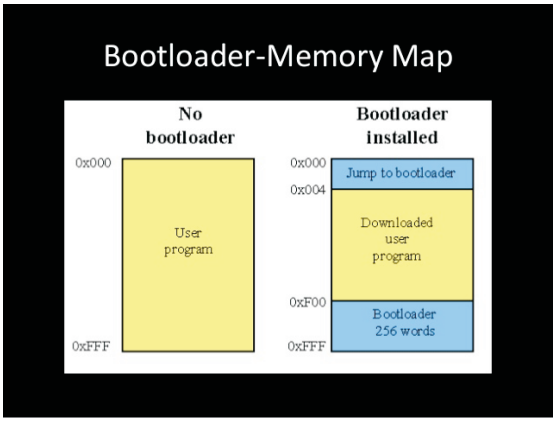
Atmel-8271J-AVR- ATmega-Datasheet_11/2015

Atmel

Con i microcontrollori e con Genuino UNO ogni riga di programma si porta via un po' della poca memoria disponibile, fino ad arrivare al suo limite. Genuino UNO dispone di 32mila byte circa di memoria (32KB meno 0,5KB per il bootloader) e in questo spazio dovrà stare tutto il tuo programma.

Con così poco spazio a disposizione, una progettazione attenta dell'applicazione per trovare una giusta media fra prestazioni e dimensioni del programma fa una significativa differenza. Certi progetti sono possibili solo se il codice è limato e ottimizzato fino all'ultimo, proprio per riuscire a stare nei 32KB disponibili.

Sopra: una pagina del datasheet del microcontrollore ATmega col parte del set di istruzioni; sotto la mappa di memoria Flash con il bootloader.



È ora di imparare l'arduinese

Grazie al lavoro fatto dal team di sviluppo di Arduino.cc, il linguaggio che avete a disposizione per programmare è una versione semplificata di C/C++, arricchito da una serie di comandi creati apposta per darvi accesso alle funzioni specifiche della scheda.

Il C dell'Arduino Software (IDE) non è molto distante - come difficoltà di apprendimento - dal Basic degli home computer di oltre trent'anni fa, ma può contare su tutto il percorso evolutivo fatto dai linguaggi di programmazione. Quello che imparerete con l'ambiente di programmazione di Arduino potrete ritrovarlo negli ambienti di sviluppo più sofisticati, anche in quello delle app per smartphone.

Come tutti gli altri linguaggi, anche questo permette di descrivere e strutturare il diagramma di flusso - che avrete sviluppato precedentemente - in una serie di istruzioni che possono essere a loro volta "compilate" in un ulteriore linguaggio – o codice oggetto - che può essere finalmente trasferito sul microcontrollore per la sua esecuzione.

Può sembrare un giro complicato, ma è proprio grazie a questi passaggi – innescati da un solo tasto dell'IDE e che avvengono senza che se ne percepisca la complessità –

2011-2016 CC-BY-NC-ND

che la programmazione dei microcontrollore delle schede Genuino è alla portata anche di chi non ha mai scritto un programma.
Come accade per le lingue straniere, ci vuole molto poco per imparare a dire "buon giorno", mentre l'apprendimento vero e proprio passa dallo studio della grammatica e del vocabolario.
Non ci stancheremo di evidenziare come non serva imparare un linguaggio senza aver prima imparato ad esprimere i concetti e a strutturare il tutto in un pensiero compiuto. Per girare la frase in altro modo, non serve conoscere una lingua se non si sa in generale cosa dire.

Descrivete il flusso logico e sarete a metà dell'opera

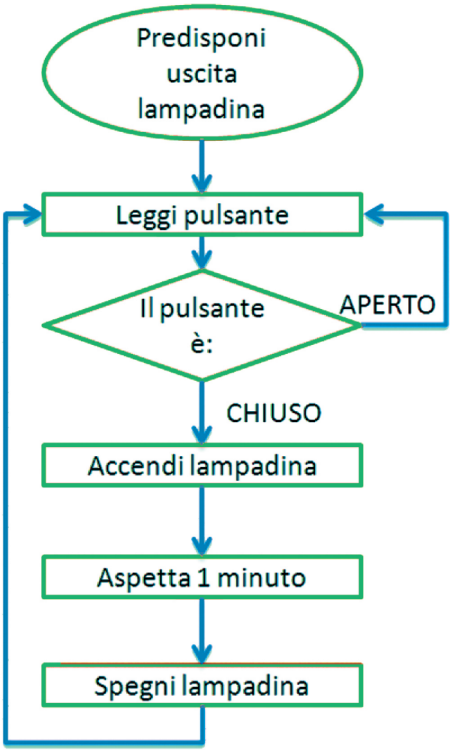
Come strutturare le idee affinché possano diventare un programma? Basta trasformarle in piccoli passi logici che descrivono funzionalmente quello che deve succedere.
I diagrammi di flusso sono di per sé una forma di programmazione e sono a un livello tale di astrazione da essere da un lato comprensibili a tutti e dall'altro sufficienti a descrivere un programma nella sua totalità. Con un diagramma di flusso ben fatto, un "programmatore" è in grado di scrivere il codice, mentre l'autore è in grado di dettagliare ciascuna funzione senza conoscere le caratteristiche specifiche del linguaggio di programmazione che verrà utilizzato.
Per fare qualche esempio, un diagramma di flusso può essere visto come una serie di frasi che descrivono passo a passo quello che succede, utilizzando delle

Le forme principali dei diagrammi di flusso

I diagrammi di flusso sono alla base di molti progetti professionali e per questo è stato fatto un serio lavoro di standardizzazione; l'argomento è anche oggetto di insegnamento in corsi di istruzione. In questo box abbiamo riportato le forme più comuni, che potrete utilizzare per i vostri diagrammi sapendo che non saranno chiari solo a voi, ma anche a tutti coloro che hanno affrontato l'argomento per hobby o per professione.

SIMBOLO	SIGNIFICATO
INIZIO / FINE	Inizio e fine della sequenza di istruzioni
	Inserimento ed emissione dei dati
	Istruzione da eseguire
	Istruzione che implica una scelta tra due possibili percorsi a seconda della valutazione di una certa condizione.

2011-2016 CC-BY-NC-ND



Il diagramma di flusso qui a sinistra descrive il processo logico e la sequenza di azioni da svolgere per gestire la luce di una scala in un condominio.

strutture grafiche predefinite per evidenziare gli elementi tipici della programmazione quali i salti condizionali, i loop (la ripetizione di una serie di istruzioni), l'ingresso e l'uscita di informazioni.
Un flusso può quindi essere basato su una "storia" come quella di una luce di temporizzata sulle scale: predispongo tutto, leggo se un tasto è chiuso (premuto) o aperto (rilasciato), se è chiuso passo alla fase successiva, altrimenti aspetto. Se l'interruttore è chiuso allora cambio stato alla lampadina aspettando 1 minuto, quindi torno a controllare se l'interruttore è premuto ricominciando da capo.
Guardando lo schema della pagina precedente si capisce sia cosa si debba fare per realizzare il programma, sia cosa il programma stesso faccia nel suo complesso. Due risultati molto importanti che dimostrano quanto l'ideazione del programma non sia nella scrittura del codice, ma nella creazione di un diagramma di flusso dettagliato e logicamente completo.
Non possiamo dire "funzionante" per un diagramma di flusso, ma è chiaro che la completezza dal punto di vista logico presuppone che chi lo ha ideato si sia posto una serie di domande alle quali ha dato una risposta esaustiva. Per esempio "Cosa succede se premo il pulsante quando la lampadina è già accesa?". Le risposte possono essere solo due: faccio qualcosa o non faccio nulla e per il primo caso posso scegliere di spegnere o prolungare. A voi la scelta, ma l'importante è pensarci. Ovviamente in base a questi ragionamenti cambia il diagramma di flusso – e conseguentemente anche il programma – aumentando ad ogni "domanda/risposta" la complessità.

Un po' di studio è inevitabile – le variabili

Anche se per gli scopi di questo libro non è necessario spiegarvi e documentarvi tutte le funzioni del linguaggio di programmazione usato nell'Arduino Software (IDE), dobbiamo quantomeno darvi i fondamenti con i quali potrete utilizzare la documentazione di riferimento inclusa con il software, accessibile dal menu Aiuto -> Guida di riferimento.

2011-2016 CC-BY-NC-ND

Come avrete intuito provando a seguire un diagramma di flusso, è necessario tener traccia di numeri e valori durante l'intero processo. Il calcolatore svolge operazioni matematiche, acquisisce valori e svolge delle "attività" in base al confronto fra valori e stati logici.
Adesso è necessario fare un piccolo sforzo e ricordare l'algebra imparta a scuola per trovare rapidamente delle analogie e comprendere il concetto di variabili.

Espressioni come "x=5" o "y=3" e "z=x-y" ci dicevano che una cerca X assumeva il valore 5, mentre Y valeva 3 e quindi Z aveva come valore 5-3, ovvero 2. Cambiando i valori di X e Y il risultato cambiava di conseguenza. Ci avevano insegnato le equazioni e ci si era aperto il mondo delle formule che esprimevano leggi fisiche o funzioni matematiche.

Con i programmi ritroviamo ancora questo tipo di notazione matematica con le variabili, alle quali però è necessario aggiungere una ulteriore dimensione: il tipo di dato. Il microprocessore deve infatti sapere con cosa ha che fare per trattare la variabile nel modo corretto. Nel caso del linguaggio di programmazione di Arduino le variabili possono essere degli interi (int), dei numeri con virgola (float), degli stati logici (bool), dei singoli caratteri (char), dei singoli byte (byte) ed altro ancora dato che ciascuno dei tipi citati può avere delle varianti per ottimizzare la quantità di memoria e il carico di lavoro sul microprocessore.

```
1 int x1;                // "x1" è una variabile intera
2 float rosso3;          // "rosso3" è una variabile con decimali
3 bool stato = 0;        // "stato" è uno stato logico pari a 0
4 char lettera = "A";    // "lettera" è il singolo carattere "A"
5 byte dato5;            // "dato5" è una variabile a singolo byte
```

Osservando gli esempi avrete notato che le etichette testuali sono lunghe a piacere e possono essere anche esplicative; l'importante è che non siano uguali alle parole riservate alle funzioni e alle istruzioni.
Nel programma useremo quindi delle parole che il programma interpreterà come il valore ad esse assegnato e anche in questo caso è importante non usare lettere a caso, ma etichette in grado di semplificare la lettura del codice.
Molto importante è ricordare che i caratteri minuscoli e maiuscoli sono gestiti come diversi fra loro, quindi "NOME" e "nome" sono due variabili diverse; spesso durante la digitazione capita di usare o dimenticare una lettera maiuscola ed avere una segnalazione d'errore come "variabile non definita" durante la compilazione.
Le variabili sono poi utilizzate con gli operatori che, come in matematica, ci permettono di fare somme, sottrazioni, moltiplicazioni, confronti, operazioni logiche e così via.

 2011-2016 CC-BY-NC-ND

```
bool Primo;
int arrivo;
int record = 130;
…                           // il programma procede ed esegue altre istruzioni
Primo = record > arrivo;    // se il numero di secondi contenuto in "arrivo"
                            // è minore di quello definito come "record" la
                            // variabile logica "Primo" è vera (vale 1)
```

Questa breve infarinatura sulle variabili vi serve solo per avere un orientamento iniziale e potrai approfondire il tema nella guida di riferimento che è accessibile su web tramite l'IDE o direttamente su *www.arduino.cc*.

Le strutture di controllo

Le strutture di controllo sono il cuore del processo logico di un programma e permettono di definire come questo procede in base ai risultati di operazioni sulle variabili o in conseguenza di dati in ingresso.
Con Arduino UNO abbiamo quindi delle parole che sono riservate alla creazione delle strutture di controllo e pertanto non possono essere utilizzate per variabili ed etichette.

if /else– La struttura più classica della programmazione; in base al risultato dell'espressione contenuta fra parentesi esegue due blocchi diversi di codice. Se l'espressione è vera verrà eseguito il codice che segue; se falsa verrà eseguito tutto il codice che segue **else. If** può anche essere utilizzato senza essere seguito da else.

```
if (pinFiveInput < 500)    // se la variabile ha un valore minore di 500
{                          // esegue l'azione A
  // azione A;
}
else                       // se la variabile è maggiore o uguale a 500
{                          // esegue l'azione B
  // azione B;
}
```

Oppure

```
if (unaVariabile > 50)     // solo se la variabile è maggiore di 50 esegue il
{                          //codice incluso fra le parentesi
  // fai qualcosa;
}
```

for – La struttura per creare un ciclo che ripete il codice un numero predefinito di volte.

```
for (int i = 0; i <= 255; i++) { // ripete per "i" che va da 0 a 255 a step di 1
  analogWrite(PWMpin, i);        // scrive il valore di "i" sul pin "PWMpin"
  delay(10);                     // si ferma per 10 millisecondi
}
```


2011-2016 CC-BY-NC-ND

switch / case – Quando ci sono varie strade da prendere in base al valore di una variabile, si può usare **switch (*variabile*)**. Per ogni valore da gestire si usa **case** *valore*: seguito dal codice da eseguire in quello specifico caso. Quando viene eseguito il codice specifico per un "case", l'esecuzione prosegue al test per il caso successivo a meno che non si usi l'istruzione **break**; che fa saltare tutti i test successivi e fa proseguire il codice dall'istruzione successiva al blocco **switch**.

```
switch (var) {
case 1:
  //fai qualcosa quando "var" è uguale a 1
  break;
case 2:
  // fai qualcosa quando "var" è uguale a 2
  break;
default:
  // se non si sono verificati i valori previsti, svolge il codice indicato
  // default è opzionale
}
```

break – permette di uscire dai cicli **do**, **for** o **while** e continuare ad eseguire codice fuori dal ciclo. Viene utilizzato anche per uscire dalla sequenza dopo il verificarsi di un **case** nella funzione **switch**.

```
for (x = 0; x < 255; x ++)
{
  digitalWrite(PWMpin, x);
  sens = analogRead(sensorPin);
  if (sens > threshold) {       // controlla il valore del sensore che se
    x = 0;                      // è superiore a threshold determina
    break;                      // l'uscita dal ciclo in corso
  }
  delay(50);
}
```

while – continua a ripetere all'infinito il codice all'interno del suo blocco finché l'espressione indicata diventa falsa. Ovviamente deve poter cambiare qualche variabile dell'espressione testata perché ci sia possibilità di uscire dalla struttura. Ad esempio si può controllare un sensore o una variabile che viene modificata dal codice all'interno della struttura.

```
sens = 0;
while (sens < 200) {
         // fa qualcosa per tutto il tempo in cui il valore letto dalla fotoresistenza
         // è inferiore a 200
  sens = AnalogRead(fotoresistenza);
}
```


2011-2016 CC-BY-NC-ND

do…while – si comporta come **while**, ma poiché il test per la ripetizione del ciclo è alla fine del blocco, il codice in esso contenuto è eseguito comunque almeno una volta.

```
do
{
  delay(50);            // attendi che i sensori si stabilizzino
  x = readSensors();    // leggi il valore dei sensori
} while (x < 100);      // continua se il valore è inferiore a 100
```

Continue – permette di far proseguire un ciclo **do**, **for** o **while** saltando le istruzioni presenti prima della fine della struttura. Si può utilizzare come parte di un test condizionale (**if**).

```
for (x = 0; x < 255; x ++)
{
  if (x > 40 && x < 120) {    // permette di saltare le istruzioni di cambio
    continue;                 // del PWM per i valori fra 41 e 119
  }
  digitalWrite(PWMpin, x);
  delay(50);
}
```

goto – permette di trasferire l'esecuzione del programma al punto definito con una etichetta. In C non si dovrebbe utilizzare questa struttura di controllo, ma in qualche caso specifico più essere di grande utilità.

return – interrompe lo svolgimento di una funzione e restituisce un risultato alla funzione che l'ha chiamata.

```
int checkSensor() {           // la funzione "checkSensor" confronta il
  if (analogRead(0) > 400) {  // valore del sensore con un valore soglia e
    return 1;                 // restituisce 1 se la soglia è stata superata
  }
  else {                      // oppure
    return 0;                 // restituisce 0 se la soglia non è superata
  }
}
```

La struttura base degli sketch

Ogni programma – o sketch – scritto per Arduino ha una struttura precisa: all'inizio ci sono gli eventuali "#INCLUDE" seguiti dalle dichiarazioni delle costanti e delle

2011-2016 CC-BY-NC-ND

variabili – messe all'inizio anche per questioni di leggibilità -, poi c'è la struttura setup() e infine c'è la struttura loop() che viene eseguita come programma e viene ripetuta se necessario anche all'infinito.

Quando si avvia una scheda Genuino dandole un reset o l'alimentazione, questa predispone le variabili in RAM come da dichiarazioni all'inizio dello sketch, quindi esegue una sola volta il setup - void setup() - e poi passa alla struttura del programma vero e proprio - void loop().

Il termine void – che è una parola riservata – significa in italiano "nulla" e indica il fatto che la struttura non restituisce alcun valore, altrimenti se avesse restituito un valore sarebbe stato necessario definire il tipo di valore (intero, virgola mobile, logico ecc.).

Ecco quindi un intero sketch di esempio:

```
1  int buttonPin = 3;      // definisco che il pin del pulsante è il 3;
2                          // se sposto il filo del pulsante posso
3                          // sistemare il software modificando solo
4                          // questa dichiarazione.
5  void setup()
6  {
7    Serial.begin(9600);          // configuro la porta seriale a 9600 baud
8    pinMode(buttonPin, INPUT);   //definisco che il pin scelto è in ingresso
9  }
10
11 void loop()
12 {
13   // ...                 // svolgo il programma vero e proprio
14 }
```

Il minimo indispensabile per uno sketch è quindi composto dalle due strutture setup () e loop() come anche indicato nel file di esempio "BareMinimum".

Le funzioni

La differenza che c'è fra un sistema di sviluppo software generico e quello per le schede Genuino è anche nelle funzioni. In ogni struttura di controllo posso eseguire dei test, delle espressioni matematiche e gestire l'interazione, ma è soprattutto grazie alle funzioni proprie dell'hardware e software delle schede Genuino che passo dal mondo del software a quello dell'hardware. L'accesso tramite funzioni alle risorse hardware del microcontrollore AVR, sia in ingresso che in uscita, concretizza il concetto di "physical computing" e mette a disposizione anche dei neofiti un

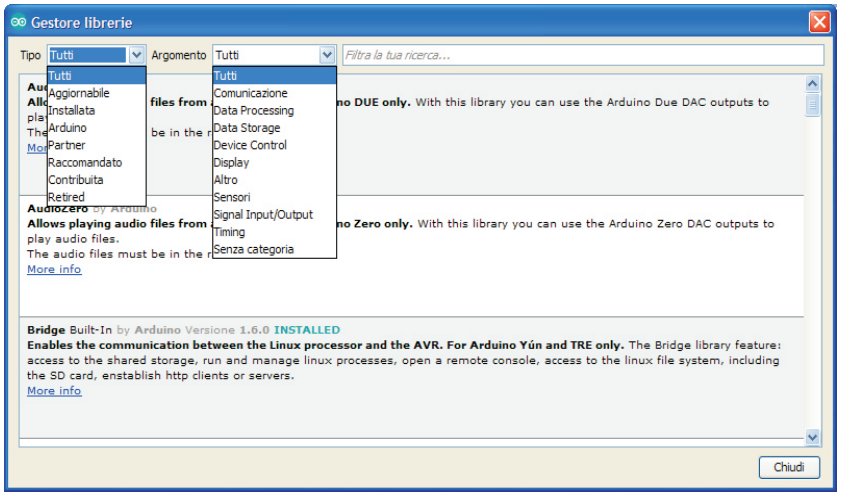
 2011-2016 CC-BY-NC-ND

insieme di funzionalità sofisticate per interagire con il mondo esterno (o fisico, come hanno voluto specificare gli ideatori).
Oltre alle funzioni più tradizionali come quelle relative alla matematica, alla trigonometria e alla matematica binaria, troviamo quindi funzioni per l'input output digitale e analogico con vari livelli di sofisticazione. Ciascuna funzione ha dei parametri e per questo non possiamo generalizzarne la descrizione.
Funzionalmente avete di fronte qualcosa che vi permette di leggere e scrivere su ciascun pin disponibile sulla scheda hardware. Se i pin sono digitali potrete portarli a uno stato logico 1 (5V) o 0 (0V), mentre se sono analogici (PWM) potrete scrivere un valore compreso in un intervallo da 0 a 255 ottenendo, come vedremo più avanti, l'equivalente di 255 scalini da 0 a 5V. Allo stesso modo potremo leggere in ingresso un valore binario o uno analogico: il primo potrà essere 0 o 1, mentre il secondo potrà variare da 0 a 1023 (un intero a 10 bit).
La guida di riferimento dell'Arduino Software (IDE) è indispensabile per trovare tutti i dettagli relativi a ciascuna delle funzioni disponibili, mentre abbiamo preparato un box in questo capitolo con gli elenchi delle funzioni raggruppate per tipo.

Fra le recenti migliorie del software c'è il Gestore Librerie grazie al quale è molto più semplice trovare le librerie relative a qualche funzione o dispositivo; sempre con il Gestore Librerie possiamo tenere le librerie scaricate alla versione più recente.

Le librerie: funzioni solo quando servono

Abbiamo detto che la memoria disponibile per il programma non è che di qualche decina di migliaia di byte e per questo il codice deve essere scritto senza sprechi. Per evitare di occupare codice per nulla, con i microcontrollori – e quindi anche con le schede Genuino – si utilizzano le librerie da includere solo quando necessario. Con le librerie si rendono disponibili nuove funzioni, legate all'hardware standard di Genuino UNO, che non sono necessarie con i normali sketch, o meglio, sono dedicate a specifici tipi di progetto e per questo possono essere utilizzate solo quando necessario.



Immaginate che la memoria di programma della vostra scheda Genuino sia come una valigia di un volo low cost: piccola e dimensionalmente ben definita. Prepararla nel modo opportuno richiede necessariamente delle scelte. Il minimo indispensabile per vestirsi e lavarsi ci deve stare, quindi con lo spazio che rimane porteremo quello che si addice al viaggio: vestiti pesanti per la montagna, costume, maschera e creme solari per il mare. Se invece dobbiamo stare via molto tempo e fare molte

2011-2016 CC-BY-NC-ND

cose, dobbiamo per forza ricorrere ad un bagaglio più grande, ovvero a una scheda con più risorse. Tornando a Genuino UNO, abbiamo a disposizione una serie di librerie standard, da includere o meno a seconda delle nostre reali esigenze di progetto e presenti nell'Arduino Software (IDE) standard:
Oltre a queste librerie, strettamente legate alle caratteristiche della scheda Genuino UNO, nell'installazione standard si trovano altre librerie direttamente gestite dal Team che sono legate a schede diverse come la Zero, la Yùn, la 101 o la Due e per questo si trovano nell'area delle librerie principali.
Negli anni, grazie al lavoro di molti utenti della community, sono state realizzate centinaia di altre librerie con diversi livelli di specializzazione e sofisticazione. Ciascuna di esse può essere inclusa nel proprio sketch purché sia correttamente installata sul computer seguendo la procedura di caricamento da ZIP o tramite il recente Gestore Librerie. Grazie a queste librerie degli utenti, Genuino UNO è in grado di acquisire istantaneamente funzionalità che altrimenti vi avrebbero richiesto non poco impegno e studio.
L'attuale installazione dell'Arduino Software (IDE) non le comprende - anche perche sono veramente molte - ma utilizzando il Gestore Librerie che è una delle significative migliorie della versione 1.6.x, si ha a disposizione un sistema di ricerca e gestione delle librerie. Possiamo cercarle per tipologia o per parola chiave e l'elenco riportato ci indica il nome, l'autore e una breve descrizione del contenuto della libreria stessa. Sempre questo Gestore Librerie si preoccupa di controllare ad ogni avvio se sono presenti versioni aggiornate delle librerie che abbiamo nella nostra installazione. In questo caso ci offre la possibilità di scegliere quale versione vogliamo scaricare rispetto a quelle disponibili.

Le librerie base dell'Arduino Software (IDE)

- **EEPROM** – per leggere e scrivere su memorie EEPROM, che conservano i dati anche senza alimentazione.
- **Ethernet** - per collegarsi ad Internet e a una rete Ethernet attraverso Arduino Ethernet Shield.
- **Firmata** – per comunicare con il computer collegato ad Arduino second un protocollo standard.
- **GSM** – per gestire le funzionalità GPRS e GSM dello Shield GSM
- **LiquidCrystal** – per gestire i display a cristalli liquidi basati su controller HD44780 o compatibile (LCD).
- **SD** – per leggere e scrivere le schede di memoria SD.
- **Servo** – per controllare i servomotori.
- **SPI** – per dialogare con dispositive che utilizzano l'interfaccia e il protocollo Serial Peripheral Interface (SPI).
- **SoftwareSerial** – per creare una porta seriale su qualsiasi pin digitale.
- **Stepper** – per gestire i motori passo passo (stepper).
- **TFT** – per scrivere e disegnare sullo specifico display grafico TFT Arduino
- **WiFi** – per connettersi alla rete Internet tramite lo shield Arduino WiFi.
- **Wire** - per gestire la comunicazione con uno o più dispositive che impiegano la modalità di comunicazione a due fili Two Wire Interface (TWI/I2C).

2011-2016 CC-BY-NC-ND

In queste due pagine abbiamo elencato tutte le strutture, variabili e funzioni disponibili nell'Arduino Software(IDE). I colori dei titoli dei gruppi sono anche quelli che nell'editor sono utilizzati per evidenziare le parole chiave.

Strutture

setup()
loop()

Strutture di controllo

if
if...else
for
switch case
while
do... while
break
continue
return
goto

Altri elementi sintattici

; (punto e virgola)
{}(parentesi graffe)
// (commento su una sola linea)
/* */ (commento multilinea)
#define
#include

Operatori aritmetici

= (assegnazione)
+ (somma)
- (sottrazione)
* (moltiplicazione)
/ (divisione)
% (modulo)

Operatori di confronto

== (uguale a)
!= (diverso da)
< (minore di)
> (maggiore di)
<= (minore o uguale a)
>= (maggiore o uguale a)

Operatori logici

&& (and)
|| (or)
! (not)

Operatori per i puntatori

* operatore di dereferenziazione
& operatore di referenziazione

Operatori sui bit

& (and sul bit)
| (or sul bit)
^ (xor sul bit)
~ (not sul bit)
<< (shift dei bit a sinistra)
>> (shift dei bit a destra)

Operatori composti

++ (incrementa)
— (decrementa)
+= (soma composta)
-= (sottrazione composta)
*= (moltiplicazione composta)
/= (divisione composta)
%= (modulo composto)
&= (and sui bit composto)
|= (or sui bit composto)

Variabili

Costanti

HIGH | **LOW**
INPUT | **OUTPUT** | **INPUT_PULLUP**
LED_BUILTIN
true | **false**
integer constants
floating point constants

Tipi di dato

void
boolean
char
unsigned char
byte
int
unsigned int
word
long

2011-2016 CC-BY-NC-ND

unsigned long
short
float
double
string - char array
String - object
array

Conversioni

char()
byte()
int()
word()
long()
float()

Scopo delle Variabili & Qualificatori

variable scope
static
volatile
const

Utilità

sizeof()
PROGMEM

Funzioni

I/O Digitale

pinMode()
digitalWrite()
digitalRead()

I/O Analogico

analogReference()
analogRead()
analogWrite() - PWM

I/O avanzato

tone()
noTone()
shiftOut()
shiftIn()
pulseIn()

Tempo

millis()
micros()
delay()
delayMicroseconds()

Matematica

min()
max()
abs()
constrain()
map()
pow()
sqrt()

Trigonometria

sin()
cos()
tan()

Caratteri

isAlphaNumeric()
isAlpha()
isAscii()
isWhitespace()
isControl()
isDigit()
isGraph()
isLowerCase()
isPrintable()
isPunct()
isSpace()
isUpperCase()
isHexadecimalDigit()

Numeri Random (casuali)

randomSeed()
random()

Bit e Bytes

lowByte()
highByte()
bitRead()
bitWrite()
bitSet()
bitClear()
bit()

Interrupt Esterni

attachInterrupt()
detachInterrupt()

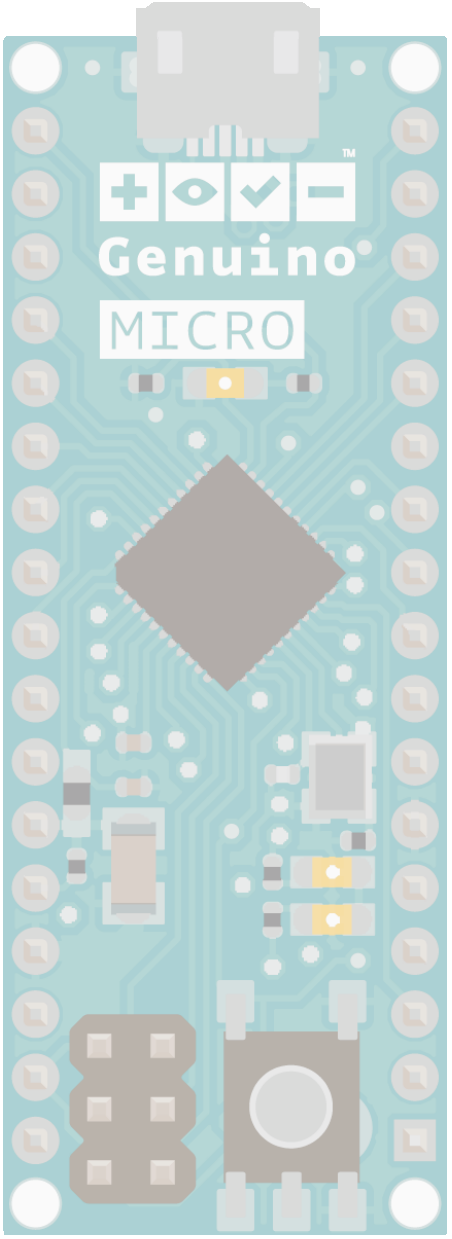
Interrupt

interrupts()
noInterrupts()

Comunicazione

Serial
Stream

2011-2016 CC-BY-NC-ND



credit: Arduino.cc

2011-2016 CC-BY-NC-ND

Pensato per essere collegato a qualcosa

La scheda Genuino UNO nasce per essere collegata facilmente a tantissime cose, talmente tante che ad oggi nessuno le ha ancora individuate tutte

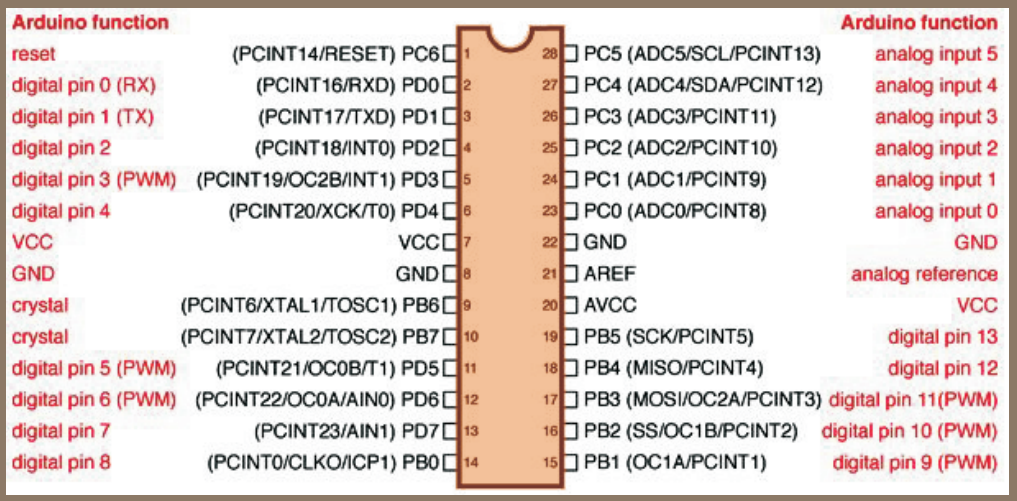
Rispetto al mondo dei personal computer, Genuino UNO - come i suoi predecessori - ha una marcia in più: quella di poter gestire in ingresso e in uscita segnali non solo digitali, ma anche analogici con grande immediatezza e semplicità grazie al software di programmazione e alle funzioni disponibili. Con queste possibilità, diventa quasi una passeggiata collegare a Genuino UNO praticamente qualsiasi cosa che sia "compatibile" con l'elettricità. Ci sono ovviamente dei limiti e delle modalità di collegamento per ogni dispositivo ed è evidente che il funzionamento a bassa tensione della scheda non permetta di pilotare direttamente dei dispositivi di potenza (lampade, motori e tutto quello che ha bisogno di più di 5V), ma con l'elettronica e l'elettricità ci sono sempre delle soluzioni per risolvere questo tipo di problematiche ricorrendo a qualche componente aggiuntivo e un circuito accessorio specifico.

Input e Output, Analogico e Digitale: le quattro parole chiave

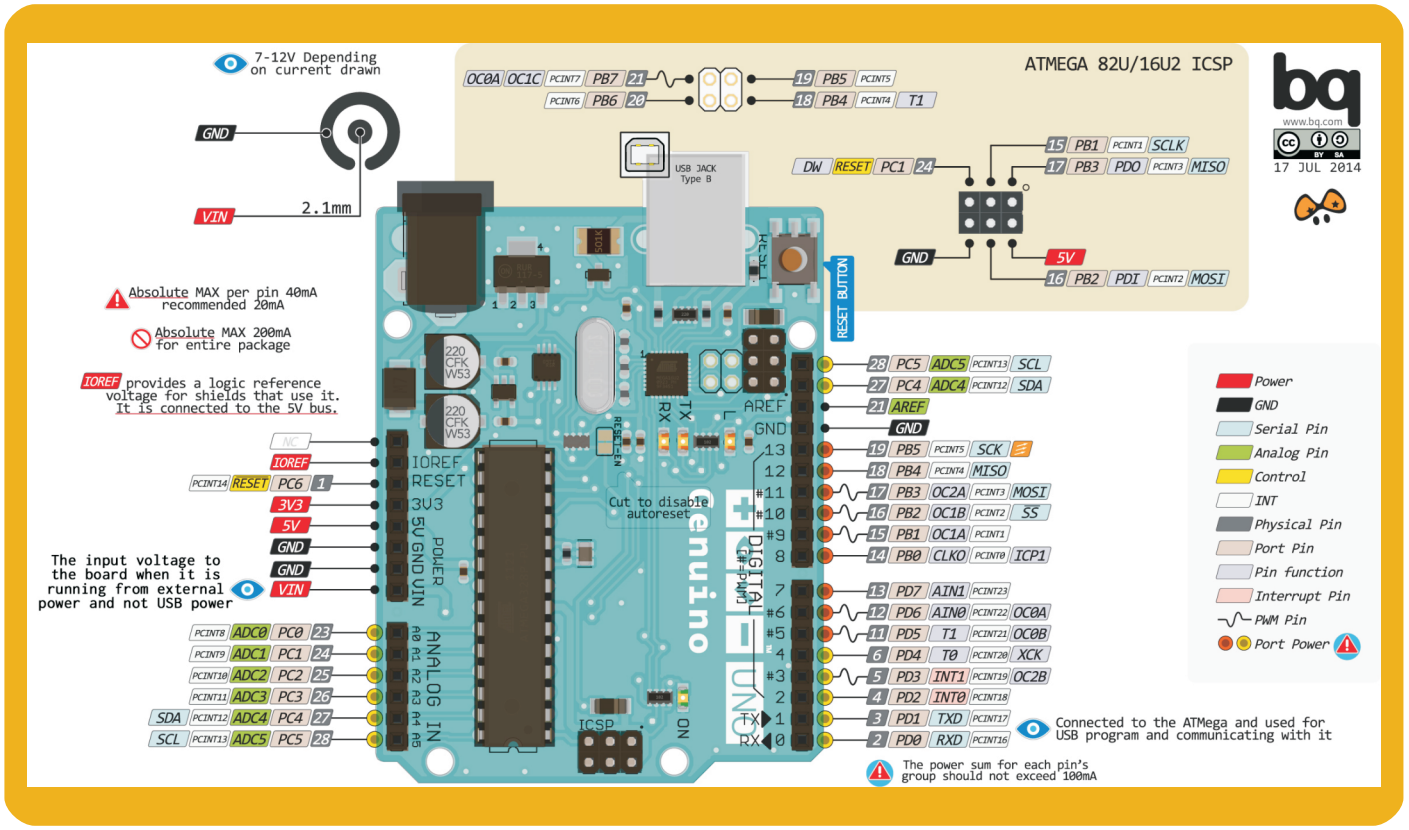
Se da una parte la flessibilità è uno degli obiettivi dei progettisti di microcontrollori, dall'altra la poca chiarezza può diventare un incubo per il neofita. Il microcontrollore Atmel scelto per Genuino UNO - il tipo ATmega328P - brilla per flessibilità, ma

Come i pin di ATmega328 diventano pin di Genuino UNO.

Per capire come il lavoro del team di sviluppo abbia semplificato la vita di chi non è un esperto di microcontrollori, basta guardare come ciascun pin del microcontrollore ATmega328 sia stato gestito assegnandogli una funzione specifica. Questo non impedisce di utilizzare i vari pin anche nelle altre modalità, ma permette di avere all'inizio un quadro semplificato e più accessibile.



2011-2016 CC-BY-NC-ND

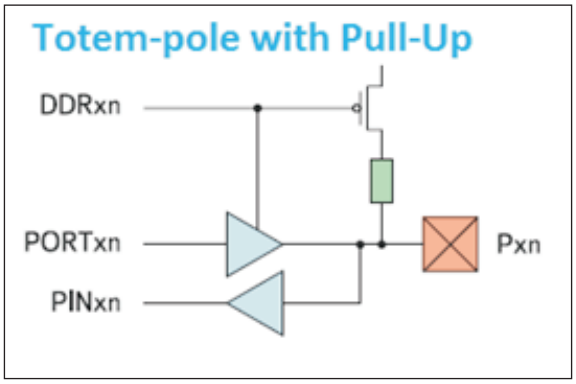


Qui sopra uno schema molto dettagliato di come i singoli pin della scheda siano collegati al microcontrollore o abbiano altre funzioni specifiche. Sotto lo schema delle porte di I/O digitali con la resistenza di pull-up attivabile via software.

allo stesso tempo potrebbe sollevare qualche perplessità in chi non ha grande esperienza nell'utilizzo di questi dispositivi. Anche questa volta, quindi, le scelte sono state fatte già dal team di sviluppo e la scheda si presenta con una ordinata serie di pin che dichiarano in modo evidente la loro funzione.
Le tipologie di collegamento che ci offre Genuino UNO sono ingresso e uscita, per segnali digitali o analogici.
Mentre le definizioni di ingresso e uscita sono chiare, dal punto di vista tecnico è il caso di spendere qualche parola per chiarire di cosa si tratti.

I/O Digitale

I pin digitali sono progettati per essere bidirezionali, ovvero in grado di passare dalla modalità di ingresso a quella di uscita attraverso la scrittura di opportuni valori nei registri di controllo (aree di memoria speciali interne al microcontrollore). Quando sono in modalità Output, possono assumere non solo valori logici 0 e 1 - corrispondenti a 0 volt e 5 volt, ma anche un terzo stato (tri-state) ad alta impedenza che di fatto li rende "ininfluenti" per qualsiasi circuito logico collegato. Non sono né 0 né 1 e allo stesso tempo non influiscono sullo stato di altre porte in uscita collegate. Questo stato risulta utile quando si vogliono collegare più dispositivi con un unico filo comune (o bus) per scambiare informazioni: chi vuole ricevere mette il pin in ingresso, chi vuole comunicare trasmette sul pin in modalità output e chi non deve far nulla va in modalità tri-state rendendosi neutro.

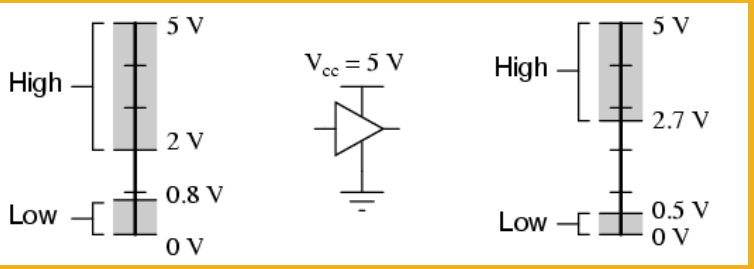


In ingresso, con le porte ad alta impedenza, basta poco perché venga rilevato il livello logico 1 e per evitare incertezze il microcontrollore dispone di resistenze di *pull-up*, attivabili su ciascun pin via software. Se non è collegato nulla, la resi-

2011-2016 CC-BY-NC-ND

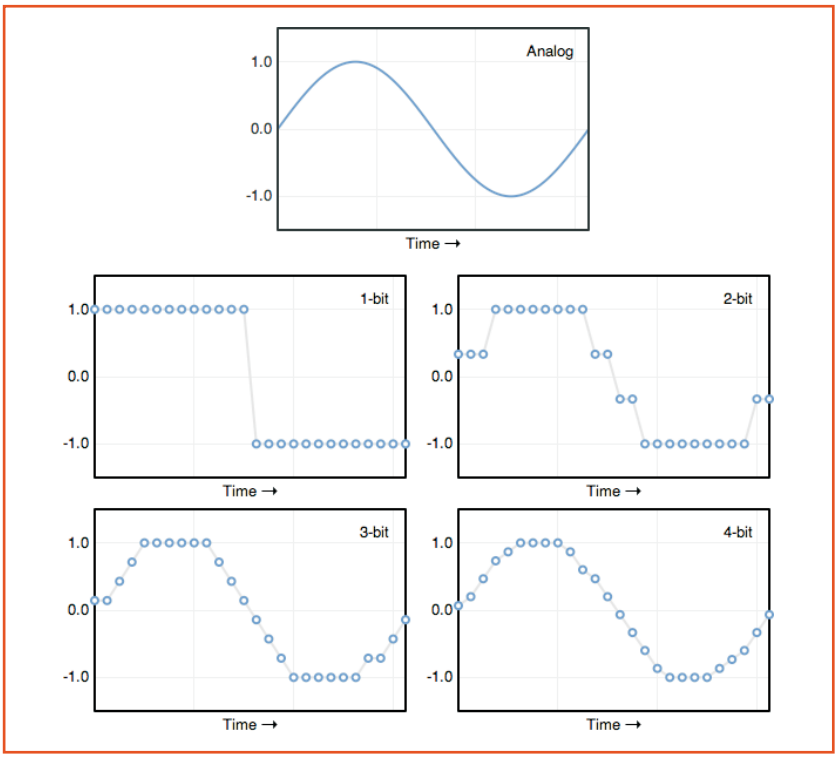
I livelli logici TTL

L'acronimo TTL - Transistor-Transistor Logic - definisce la tensione dei segnali utilizzati per comunicare. Un altro tipo di circuito, il CMOS, utilizza livelli diversi. Nel mondo digitale, però, non abbiamo a che fare con valori esatti e sempre come da specifiche e per questo sono state definite le aree entro le quali una tensione viene considerata come LOW (0), HIGH (1) o non definita. Nello schema qui sotto sono riportati i valori ritenuti accettabili; a sinistra i valori in ingresso e a sinistra quelli in uscita.



stenza di pull-up (una resistenza da circa 35k ohm verso il positivo) porta a 1 il valore letto, senza però che i 35k ohm possano creare problemi nel caso di un livello logico 1 generato da un dispositivo collegato.
Dal punto di vista squisitamente elettronico, i pin configurati in ingresso non rappresentano un carico di cui preoccuparsi e quindi anche sensori e dispositivi a bassa potenza possono generare i livelli logici necessari. Da 0 a 2,5 volt avremo una lettura di uno stato logico 0, mentre oltre 2,5 volt fino a 5 volt saranno considerati come stato logico 1. La tensione è ovviamente riferito alla massa.
In uscita, i pin di Genuino UNO hanno dei circuiti di pilotaggio definiti come simmetrici, ovvero in grado di generare dei livelli 0 e 1 non solo con carichi ad alta impedenza (e quindi bassissimo assorbimento), ma anche su carichi che assorbono un massimo di 40 mA, fino a raggiungere un totale complessivo per tutti i vari pin di 150 mA. Ogni pin digitale non dovrebbe essere fatto lavorare con assorbimenti superiori a 20 mA per non sottoporre il microcontrollore a carichi fuori dalle specifiche di funzionamento ottimali.
In pratica, potete collegare direttamente qualche LED, un buzzer e altri dispositivi senza problemi. La corrente è anche sufficiente ad alimentare dei sensori, che possono quindi essere accesi e spenti in base alle necessità, economizzando il consumo energetico.

Qui sotto gli effetti del campionamento a 1, 2, 3 e 4 bit di un segnale analogico.
Da notare come il segnale venga approssimato su scalini che dipendono dal numero di bit usati.



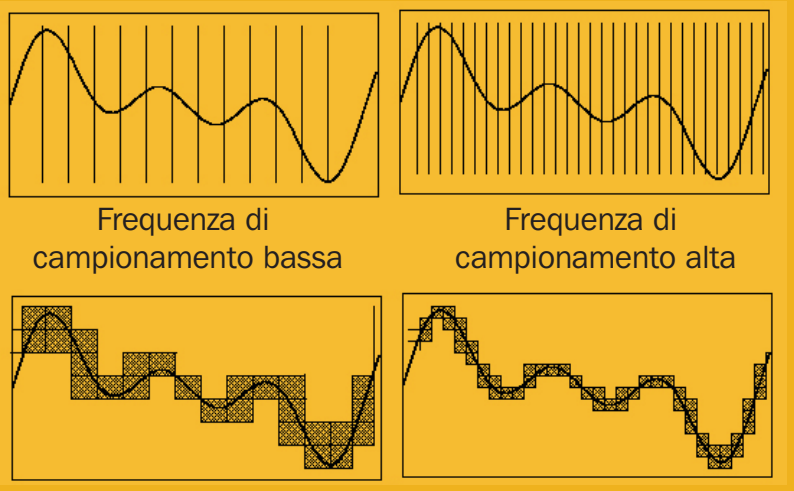
Ingresso Analogico

Il comportamento delle schede Genuino con i segnali analogici non è simmetrico fra ingresso e uscita, ovvero non possiamo generare segnali analogici con la stessa precisione che abbiamo in ingresso e il tipo di segnale analogico non è neppure un vero livello di tensio-

2011-2016 CC-BY-NC-ND

La frequenza di campionamento

Oltre al numero di bit, nella conversione da analogico a digitale conta molto anche la velocità con cui si esegue il campionamento: più questa è elevata e più sarà la possibilità di catturare in digitale le frequenze più alte grazie al miglior dettaglio sull'asse del tempo. I CD audio hanno una frequenza di campionamento di 44,100 Hz a 16 bit. I bit definiscono il dettaglio sull'asse del livello.



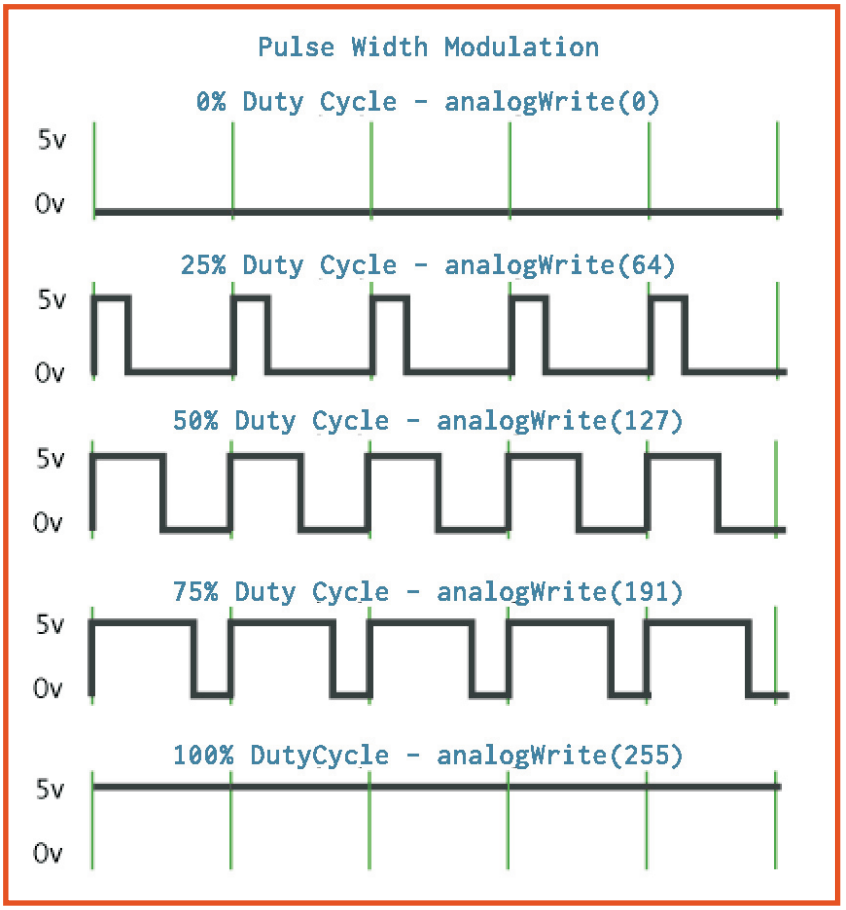
ne fra un minimo e un massimo a meno che non si usi un circuito d'integrazione.
Questo mostra chiaramente come ci siano dei limiti imposti dal costo contenuto dell'hardware, limiti ai quali si è posto un rimedio "software".
Innanzi tutto vale la pena capire meglio come un dispositivo digitale gestisca la lettura di un segnale analogico, attraverso il processo della conversione o campionatura, svolto dal convertitore analogico digitale (A/D).
Il convertitore A/D è un circuito che misura la tensione in ingresso ogni volta che viene interrogato. Si tratta di una misurazione istantanea che viene poi convertita in un valore binario. L'intervallo di tensioni misurabile viene suddiviso in tanti livelli discreti - o scalini - quanti sono i valori esprimibili con i bit a disposizione. Se il convertitore è a 8 bit, potrà esprimere 256 valori; se è a 16 bit ne esprimerà 65536. Il microcontrollore utilizzato ha un convertitore A/D a 10 bit e quindi può esprimere 1024 valori diversi.
È importante comprendere come il numero di scalini (valori) gestiti dal convertitore non abbia un rapporto diretto con l'estensione dei valori stessi: questa è infatti legata all'intervallo di misura che va da 0 alla tensione di alimentazione/riferimento del convertitore (5V per Genuino UNO). I bit di risoluzione ci permettono solo di avere una lettura più o meno precisa. In pratica, senza intervenire con circuiti accessori, con Genuino UNO possiamo leggere valori analogici fra 0 e 5 volt, con 1024 livelli discreti.
La lettura è lineare e quindi il valore 512 corrisponde a 2,5V, mentre 128 corrisponde a 0,625 volt e così via. Un singolo livello è pari a 0,0048 volt e questa di fatto è la risoluzione teorica del convertitore della nostra scheda Genuino UNO.
Oltre alla risoluzione, è anche importante la frequenza con cui la lettura viene ripetuta - o frequenza di

Frequenza di campionamento	Qualità audio	Banda passante
11.025 Hz	Radio AM di bassa qualità	0 - 5.512 Hz
22.050 Hz	Radio FM di scarsa qualità	0 - 11.050 Hz
32.000 Hz	Radio FM di buona qualità	0 - 16.000 Hz
44.100 Hz	Qualità CD Audio	0 - 22.050 Hz
48.000 Hz	Qualità DVD	0 - 24.000 Hz
96.000 Hz	Audio ad Alta Definizione DVD	0 - 48.000 Hz

2011-2016 CC-BY-NC-ND

campionamento -, specie se si vogliono leggere segnali che variano molto rapidamente, come ad esempio un segnale audio. Diciamo subito che Genuino UNO può arrivare al massimo a registrare della voce con una fedeltà limitata, mentre per la musica è necessario uno shield dedicato. La frequenza massima di lettura senza usare "trucchi" è di circa 9 kHz e questo permette di campionare frequenze fino a 4,5 kHz. Il convertitore A/D di Genuino UNO è disponibile su cinque pin e questo permette di dotare la scheda di più sensori, non necessariamente uguali fra loro, per avere una lettura del mondo esterno sotto forma di valori analogici.

Le uscite analogiche di Genuino UNO utilizzano la modulazione d'ampiezza dell'impulso (PWM) per simulare 256 livelli fra 0 e 5 volt.

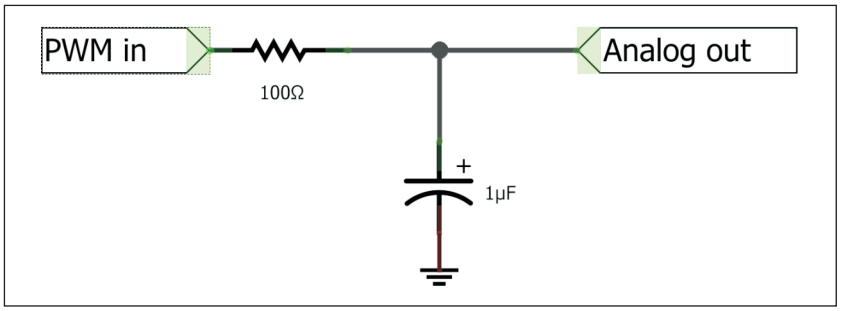


Uscita analogica - PWM

Le uscite analogiche di Arduino UNO non sono realmente in grado di produrre una tensione variabile - sempre a step discreti - da 0 a 5 V, ma sfruttano la tecnica del PWM per ottenere un effetto in molti casi equivalente. PWM è l'acronimo di Pulse Width Modulation - modulazione dell'ampiezza d'impulso - e da qualche anno è la tecnica utilizzata per modulare l'intensità dei LED.
Sostanzialmente il segnale prodotto varia solo fra i due livelli logici 0 e 1, pari a 0 e 5 V, ma il rapporto fra i livelli viene variato all'interno del singolo ciclo per simulare una variazione di tensione. Voglio fornire 0,5 V al mio dispositivo? Genero un segnale PWM con il 10% del ciclo positivo. Voglio fornire 2,5V? Allora il ciclo positivo è del 50%.
Se facciamo una media nel tempo della tensione, otterremo effettivamente una serie di livelli variabili fra 0 e 5V pari agli step disponibili per il duty cycle del PWM. Nel caso di Genuino il PWM è a 8 bit, ovvero ha 256 livelli. Nel caso specifico dei LED, il PWM determina la maggiore o minore luminosità percepita gestendo il tempo in cui il led stesso è acceso. Una volta che si supera una certa frequenza nel lampeggio, l'occhio umano non percepisce più la "vibrazione" e vede solo un LED con luminosità più o meno forte. Variando la durata del rapporto acceso/spento si ottiene quindi una luce più o meno intensa, come se si pilotasse il LED con una

 2011-2016 CC-BY-NC-ND

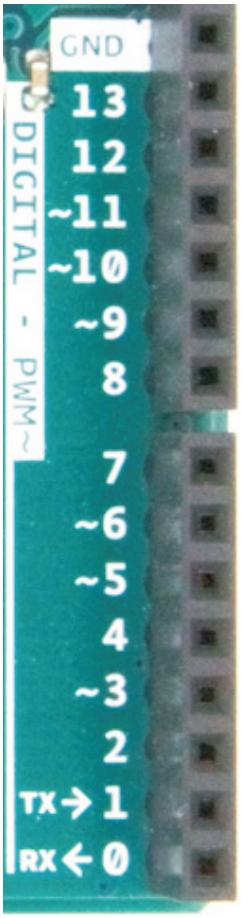
tensione variabile. Poiché la tensione varia sempre fra 0 e 5 volt, è importante ricordare che ci sono dispositivi che non sopportano bene i 5V e quindi vanno protetti con resistenze di limitazione della corrente o altri accorgimenti. Una media di 2,5V corrisponde a 5V per il 50% del tempo. In pratica, quindi, predisponete il circuito per il suo funzionamento "sicuro" con 5V, dopodiché con il PWM riducete la forza o l'intensità senza il problema di evitare la massima potenza per non bruciare il componente.



Da PWM a tensione

Se è proprio indispensabile avere una tensione e non un segnale PWM, si può usare un semplice circuito che fa la media nel tempo degli impulsi e li trasforma quindi in una vera tensione. Si tratta di un filtro passa basso composto da una resistenza e da un condensatore.

A seconda dei valori di ingresso, del carico collegato in uscita e della frequenza del segnale PWM, vanno selezionati opportunamente i valori di R e C, ricordando che comunque questo tipo di filtro ha una propria velocità di risposta alle variazioni nel segnale PWM e inoltre lascia passare come "ronzio" di fondo la portante del segnale PWM (ovvero la frequenza dell'onda quadra di cui varia il rapporto on/off).



Diamo un senso a tutte le scritte: cosa posso fare con ciascuno dei PIN della scheda

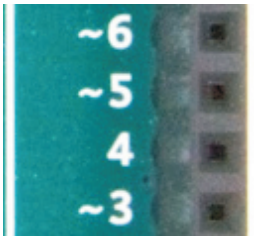
Ora che è chiaro come funzionino input, output, analogico e digitale, puoi guardare la scheda Arduino UNO con maggiore confidenza, iniziando a immaginare quello che puoi collegare.

Sul lato destro della scheda si trovano quattordici pin definiti come "**Digital**" e affiancati in alcuni casi dal segno "~". Questi pin possono essere utilizzati indifferentemente come ingressi o uscite tramite le funzioni pinMode(), digitalWrite(), e digitalRead(). Ciascun pin digitale può fornire o assorbire **20 mA** e fornisce 5V in uscita quando attivo. Come indicato in precedenza, se impostato su input, ciascun pin

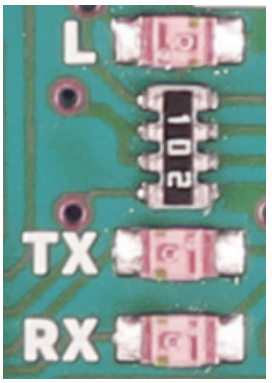
2011-2016 CC-BY-NC-ND

può essere tenuto a livello alto tramite una resistenza di pull-up. Se non viene fornito qualche comando specifico in fase di inizializzazione (setup), tutti i pin digitali sono in input e le resistenze di pull-up sono scollegate. Alcuni pin digitali hanno delle etichette di testo specifiche e svolgono opzionalmente funzioni speciali, in base allo sketch caricato e al setup.

Porta Seriale: pin 0 (**RX**) and pin 1 (TX). **Permette** di dialogare in seriale - con livelli TTL - con altri dispositivi ricevendo sul pin 0 e trasmettendo sul pin 1. I due pin sono collegati ai pin corrispondenti del chip ATmega8U2 USB-to-TTL.



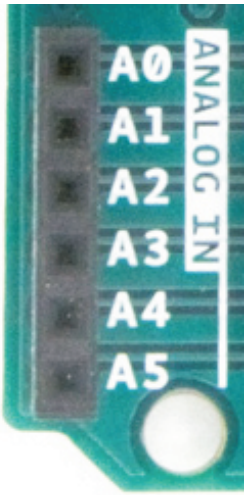
Interrupt Esterni: pin 2 e pin 3. Questi due pin permettono di attivare un interrupt - particolare situazione in cui il microcontrollore viene portato ad eseguire del codice specifico, se previsto - quando il livello subisce una variazione secondo una delle quattro modalità previste. Queste sono LOW (valore basso), CHANGE (tutte le volte che il pin digitale cambia stato), RISING (l'interrupt avviene quando il pin passa da 0 a 1) e FALLING (il pin passa da 1 a 0). La funzione con cui gestire l'interrupt è attachInterrupt().



PWM (~): pin 3, 5, 6, 9, 10, e 11. Questi pin supportano la generazione di segnali PWM direttamente gestiti dall'hardware ATmega; la frequenza dell'onda quadra generata dipende da alcuni parametri che possono essere modificati, ma di default è circa 490 Hz. Il valore del PWM varia da 0 a 255 con 8 bit di risoluzione. La funzione specifica per gestire il PWM su questi pin è analogWrite().

SPI: pin 10 (**SS**), 11 (**MOSI**), 12 (**MISO**) e 13 (**SCK**). Questi pin supportano il protocollo di comunicazione seriale sincrono SPI - Serial Peripheral Interface - che viene utilizzato per comunicazioni su brevi distanze verso una periferica o un altro microcontrollore. La gestione di SPI avviene tramite una apposita libreria inclusa nell'Arduino Software (IDE).

LED: pin 13. Dato che far lampeggiare un LED è fra le prime attività consigliate al neofita della piattaforma Arduino, il team degli sviluppatori ha pensato di inserire direttamente sulla scheda un LED collegato opportunamente al pin digitale 13. Per accenderlo basta impostare il pin 13 su HIGH (digitalWrite(13, HIGH);), mentre per spegnerlo il valore deve essere LOW (digitalWrite(13, LOW);).

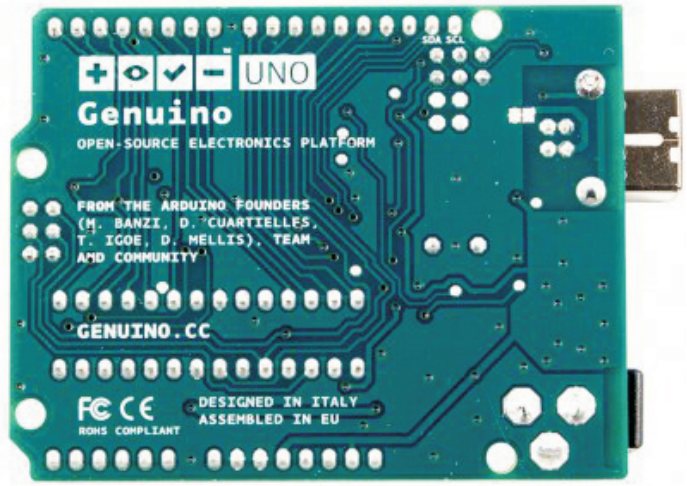


Nella parte bassa a sinistra si trovano i cinque pin di ingresso analogici, denominati A0, A1… A5. Su di essi si può leggere una tensione compresa fra 0 e 5 V riferiti alla massa della scheda. I 1024 valori - pari a 10 bit di risoluzione - sono corrispondenti all'intervallo di lettura. Per rendere il livello massimo della scala diverso

2011-2016 CC-BY-NC-ND

da 5V è possibile usare sia le sorgenti interne che una sorgente esterna collegata al pin AREF - che si trova in alto a destra, sopra il pin digitale 13 - e la funzione analogReference(). Usare un limite superiore più basso, come INTERNAL da 1,1 volt, permette di avere una maggiore risoluzione sulle variazioni di una tensione debole in quanto ogni gradino rilevabile è pari a Riferimento/1024. La tensione di riferimento NON PUO' superare i 5V o essere inferiore a 0V, altrimenti si può danneggiare il microcontrollore.

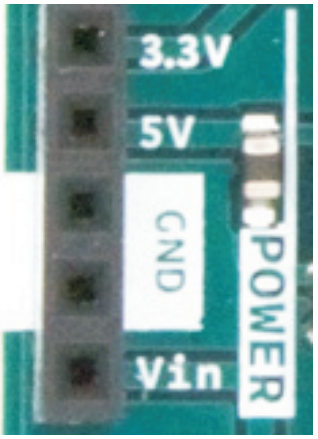
Anche su questi pin sono presenti delle funzionalità per applicazioni particolari:



TWI: pin A4 (**SDA**) e pin A5 (**SCL**) - disponibili come nuovi collegamenti sopra il pin AREF e marchiati SDA e SCL sulla parte rame della scheda Rev. 3 - per comunicare con i dispositivi I2C e TWI che supportano la comunicazione con due soli fili. Per gestire questo protocollo Genuino UNO dispone di una libreria standard dedicata, chiamata Wire.

Sempre a sinistra, sopra il connettore degli ingressi analogici si trova la serie di collegamenti legati all'alimentazione.

VIN. Genuino UNO può essere alimentato in vari modi. Quando è collegato al computer tramite USB preleva da lì i 5V, altrimenti li può ottenere da un'alimentazione fra 7 e 12 volt fornita tramite un alimentatore esterno collegato alla presa di fianco a quella USB, oppure tramite questo pin. Da notare che la presa e questo pin sono collegati fra loro e quindi se l'alimentatore esterno fornisce per esempio 9 volt, su VIN troveremo 9 volt.

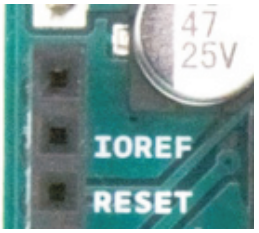


5V. L'alimentazione regolata e stabilizzata, usata per il microcontrollore e la circuiteria presente sulla scheda. Questa tensione arriva o dal regolatore presente sulla scheda a cui è fornita una tensione fra 7 e 12 volt, o dalla porta USB, alla quale si può collegare un computer che fornisce l'alimentazione o anche un semplice alimentatore con porta USB, purché di qualità.

2011-2016 CC-BY-NC-ND

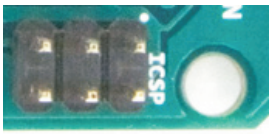
3V3. Una tensione di 3,3 volt generata dal regolatore della scheda. La corrente massima disponibile è di 50 mA.

GND. La massa della scheda (0 volt).



Reset. Il pin da mettere a massa per generare il segnale di reset, in alternativa a quello generato con il pulsante presente in alto a destra. Viene usato per montare un tastino di reset su quegli shield che per la loro forma rendono inaccessibile il tastino di reset originale.

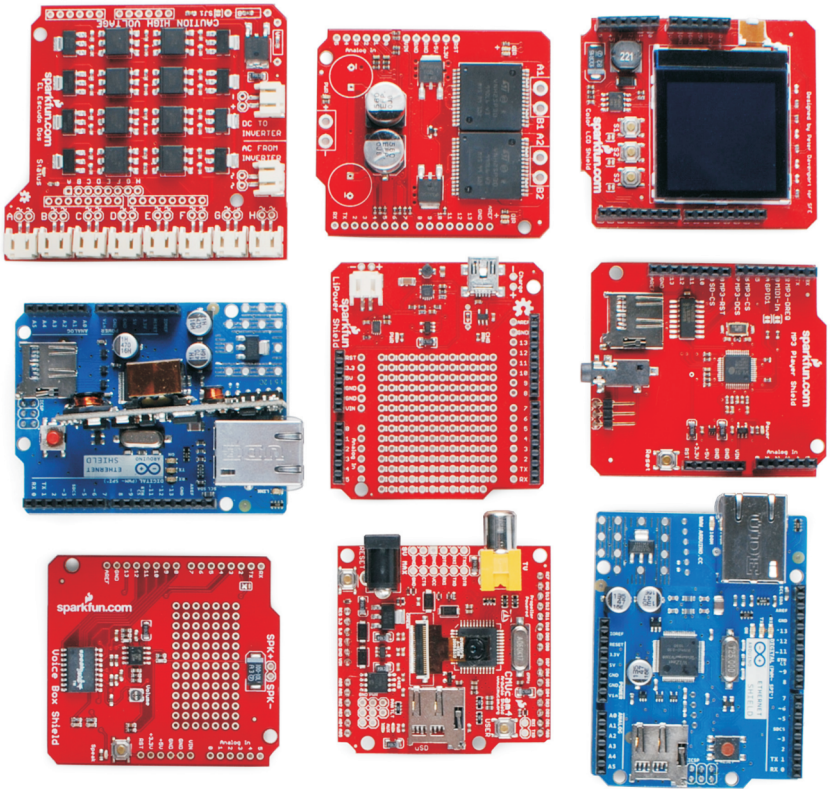
IOREF permette agli shield di adattarsi alla tensione di alimentazione fornita dalla scheda. Si tratta di un nuovo pin della Rev. 3. Sopra IOREF si trova un ulteriore nuovo pin che al momento è riservato per impieghi futuri.



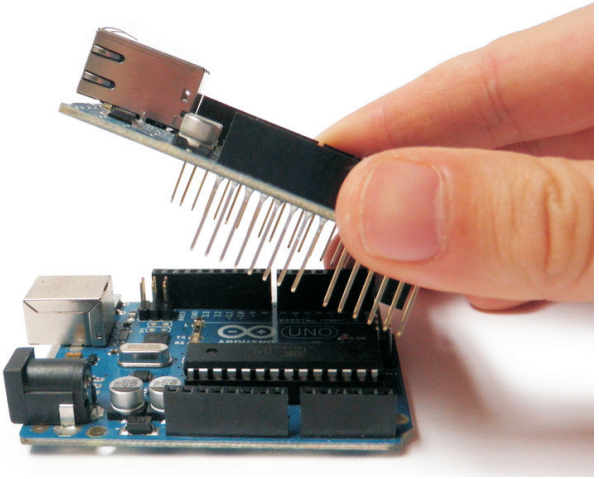
ICSP (In-Circuit Serial Programming) è un connettore da sei pin presente in centro nella parte bassa della scheda ed è utilizzabile per programmare direttamente - attraverso dell'hardware specifico - il microcontrollore.

Gli shield

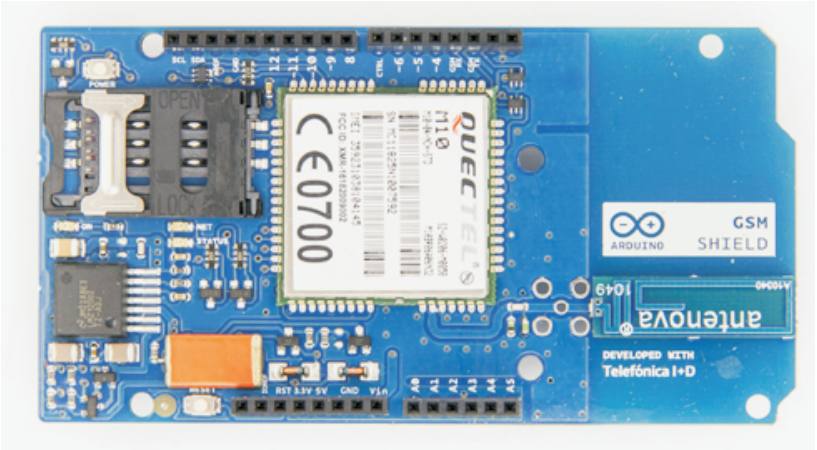
Una delle caratteristiche peculiari del progetto Arduino è la disponibilità di hardware accessorio - denominato shield - che permette con una semplicità estrema, pari a quella dei mattoncini Lego, l'espansione delle funzionalità della scheda di base.
La presenza dei connettori sui bordi della scheda Genuino UNO e la disponibilità degli schemi elettrici ha permesso a tutti coloro che ne hanno sentito il bisogno di sviluppare dell'hardware aggiuntivo con la ragionevole certezza di un collegamento immediato e la compatibilità con le varie versioni di scheda.
Tutti i vari segnali/pin che abbiamo visto nel capitolo precedente sono infatti quello che serve per aggiungere altri dispositivi mantenendone il pieno con-



2011-2016 CC-BY-NC-ND

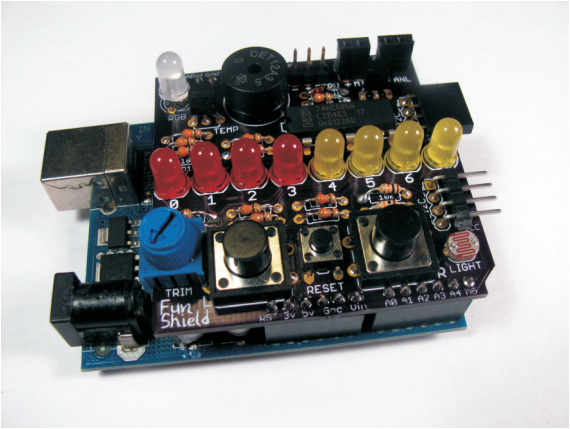


trollo. Le interfacce SPI o i vari pin digitali rendono semplice l'interfacciamento di altri chip sofisticati e così nel giro di qualche anno sono comparsi degli shield veramente interessanti.
Censirli tutti è quasi impossibile, mentre quelli più significativi e diffusi riguardano i temi della comunicazione wireless, della gestione dei motori per la robotica, dell'interfacciamento con LED e matrici, della gestione audio ad alta fedeltà, della prototipazione e della sperimentazione.
Dietro ad ogni shield c'è spesso una libreria dedicata e quindi ogni nuova caratteristica diventa disponibile al software tramite funzioni che si comportano come quelle standard che hai già imparato ad utilizzare.



Ufficiali e non

Il passaggio dalla vecchia alla nuova produzione ha creato qualche cambiamento nell'assortimento dei prodotti e quindi con il marchio Genuino, al momento in cui scriviamo (gennaio 2016), non ci sono shield in vendita, ma vale la pena ricordare quello WiFi annunciato in America - Arduino WiFi Shield 101 - che dispone di un nuovo chip con la crittografia integrata per semplificare le connessioni IoT (Internet of Things) protette e il sempreverde GSM Shield realizzato assieme a Telefonica e tutt'ora cavallo di battaglia per molti progetti.



La facilità con cui è possibile connettere sensori, attuatori e dispositivi anche di una certa complessità, assieme alla crescente integrazione di dispositivi a bordo della scheda stessa - si veda la nuova Genuino 101 - stanno un po' cambiando l'atteggiamento degli sviluppatori di hardware.
Da qualche tempo è nata la tendenza a non realizzare più uno shield, ma di integrare nell'intero circuito dedicato a una funzione specifica anche la circuiteria di Genuino. Così è successo per alcune schede per stampante 3D e per dei

2011-2016 CC-BY-NC-ND

É ora di preparare la nostra cucina tecnologica

Abbiamo capito con chi abbiamo a che fare, ora prepariamo tutti gli elementi necessari a lavorare con Genuino UNO

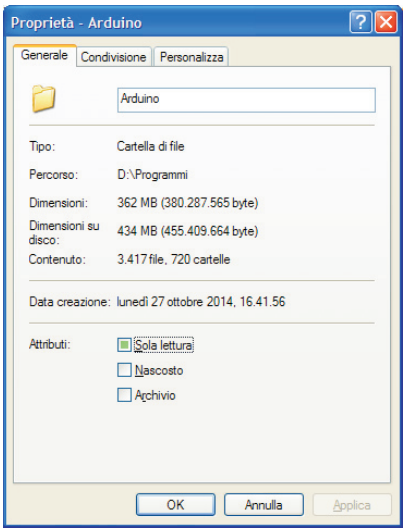
Il grande lavoro di semplificazione del progetto Arduino, sia a livello software, sia a livello hardware, ha reso veramente minimali i requisiti per utilizzare il sistema. Questi requisiti negli anni non sono poi cambiati di molto e mentre i sistemi operativi si avvicendavano nelle loro release, anche l'Arduino Software (IDE) cresceva, ma senza modificare in modo significativo le caratteristiche della piattaforma utilizzata per funzionare.

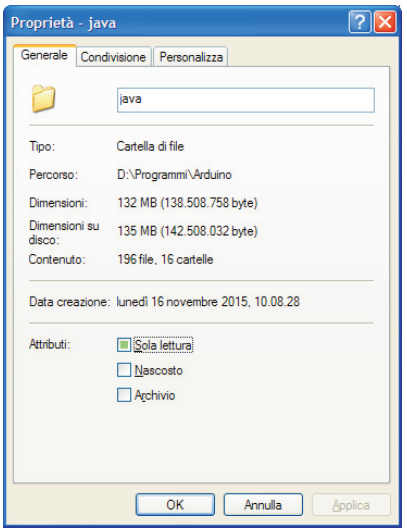
Ovviamente vi serve un personal computer che sia in grado di far funzionare applicazioni in Java, quindi vanno bene i computer

- Windows a partire da XP, Vista, Seven fino a 10;
- Mac purché abbiano Max OS X 10.3.9 o superiore;
- Linux con Ubuntu, Debian, Gentoo e molte altre distribuzioni ricordando che con Linux c'è da faticare un pochino per predisporre tutti i programmi e le librerie richiesti. Il sito Arduino.cc dispone di un'area dedicata all'installazione su Linux **qui** e vi consigliamo di partire da lì se avete intenzione di usare questo sistema operativo.

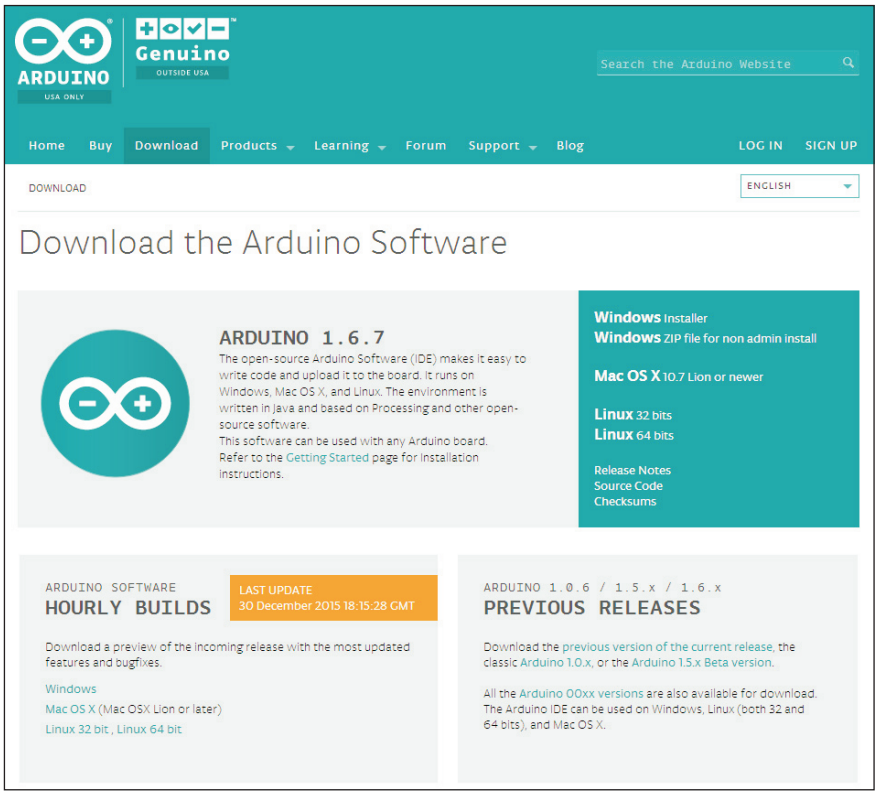
Mettiamo a punto il computer

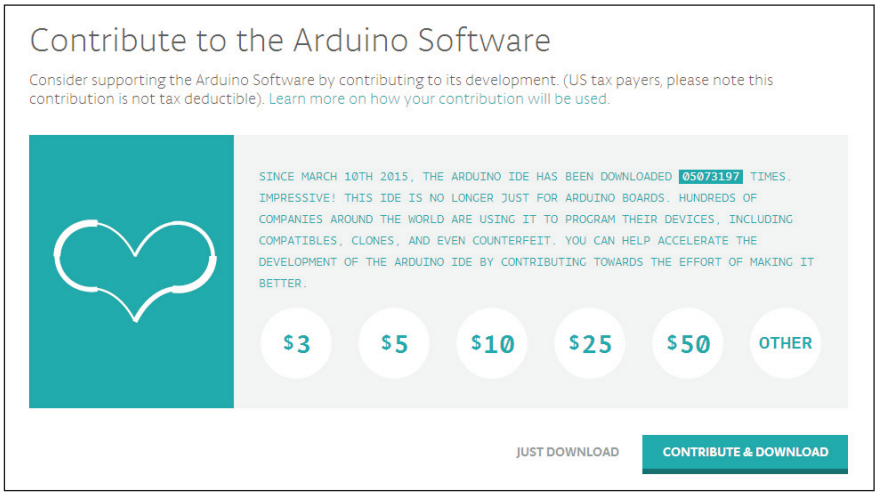
Il sistema di sviluppo è scritto in Java e il pacchetto d'installazione contiene tutto il necessario perché Java sia presente nelle specifiche cartelle all'interno della cartella principale d'installazione con il necessario per funzionare.
Dal punto di vista della memoria RAM e dello spazio su disco, Java ha dei requisiti ragionevoli, ma può essere una buona idea avere un po' di memoria RAM disponibile (almeno 512 MB) visto che durante l'esecuzione Java tende ad occuparne parecchia. In 350 MB circa si installa su disco rigido tutto l'Arduino Software (IDE) con le sue librerie. Ovviamente avere dello spazio libero su disco è indispensabile per poter lavorare e quindi qualche GB a disposizione o almeno il 20% sulle dimensioni totali dell'hard disk sarebbe da lasciare sempre libero. I file che verranno creati con i vostri sketch saranno memorizzati in una cartella all'interno della vostra cartella





2011-2016 CC-BY-NC-ND





utente, ovvero in "Documenti/Arduino/", così come le librerie scaricate tramite Gestore Librerie che abbiamo visto nel precedente capitolo. Anche gli eventuali core aggiuntivi saranno memorizzati nella vostra cartella utente.

Scarichiamo e installiamo l'IDE

Nella confezione della scheda Genuino UNO non avete trovato alcun dischetto o memoria contenente il software. Dovete obbligatoriamente andare online su internet e scaricarlo dalla pagina all'indirizzo web **www.arduino.cc/en/Main/Software**
Qui si trovano le varie versioni disponibili per le relative piatteforme supportate.
Solo con Linux si deve scegliere la versione corretta per i 32 o 64 bit, mentre su Windows e Mac OS X la versione è a 32 bit. Per Windows ci sono due versioni: con installer automatico e in formato compresso (zip) con la struttura delle cartelle pronta per essere espansa in una cartella a scelta dall'utente. Nella nota si dice che la versione Zip è per chi non ha i diritti di amministratore sul computer, ma anche l'installazione dei driver della scheda richiede i diritti di amministratore e quindi se non si è mai collegata la scheda Genuino al computer, sarà necessario installare i driver e, necessariamente, avere i diritti di amministratore. Scelta la versione avrete la possibilità di dimostrare il vostro apprezzamento per il lavoro svolto dal Team di Arduino.cc effettuando una donazione in denaro. Il software è infatti utilizzato da milioni di utenti e non è detto che questi abbiano acquistato una scheda Genuino contribuendo con una piccola royalty allo sviluppo del software e della piattaforma; con la donazione, anche chi sta utilizzando prodotti che usano l'Arduino Software (IDE) - in quanto compatibili - può dimostrare il suo apprezzamento e consentire al Team di proseguire nel suo lavoro.
Nella stessa pagina trovate anche il collegamento alle versioni precedenti e quello alle "build" orarie, ovvero alle prerelease che vengono realizzate con una elevata

2011-2016 CC-BY-NC-ND

frequenza e che recepiscono le migliorie, i fix e le correzioni via via che vengono implementate sul codice. Il processo vede la partecipazione della community con un processo di segnalazione, discussione e chiusura delle richieste su Github. Queste versioni sono però passibili di modifiche e vi consigliamo pertanto di far riferimento alle versioni stabili.
Vi consigliamo di utilizzare l'installer Windows – supponendo che abbiate una macchina iWndows - che provvede anche all'installazione dei driver richiesti dalla vostra scheda Genuino UNO. Al termine avrete sul desktop l'icona dell'Arduino Software (IDE). Se invece avete scelto la strada del file compresso, dovrete aprire la cartella in cui lo avete estratto e poi creare un collegamento con il file **Arduino.exe** sul desktop.

Colleghiamo l'hardware

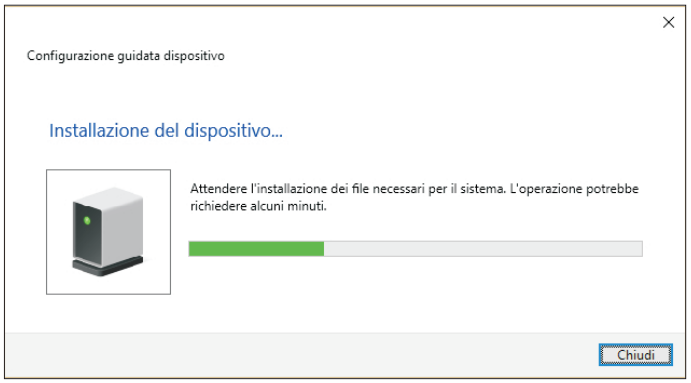
Questa fase, con il primo collegamento, ha bisogno di un po' di attenzione in quanto il computer deve installare il driver da USB a Seriale indispensabile per far pilotare la scheda dall'IDE.

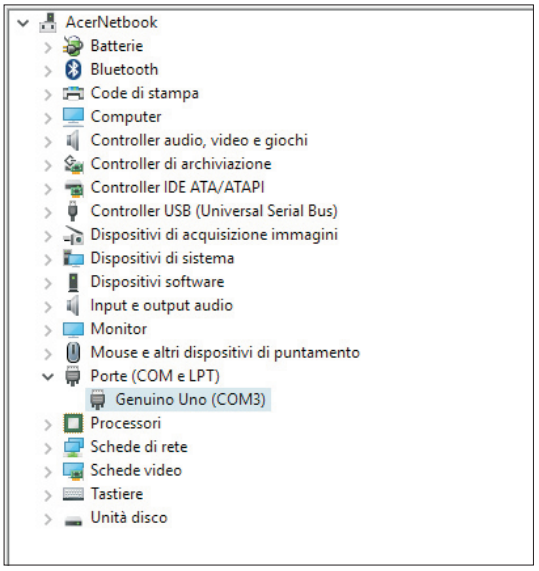
Se il vostro computer non è stato utilizzato in precedenza con schede Arduino o Genuino, è con Windows XP e ha avuto una "intensa vita" di collegamenti a periferiche e dispositivi vari, potrebbe avere qualche difficoltà a riconoscere la scheda nel modo corretto. Si tratta di una possibilità remota, ma con Windows XP e i driver USB, le sorprese sono sempre dietro l'angolo. Con Vista, Seven, 8 e 10 l'installazione non presenta difficoltà specifiche dato che i driver sono anche firmati digitalmente e non creano problemi al sistema operativo.

Innanzitutto procuratevi il cavetto di collegamento fra scheda Genuino UNO e personal computer. Si tratta di un cavetto che da una parte ha una maschio USB di Tipo A - da collegare al computer - e dall'altra un maschio USB Tipo B da collegare alla scheda.

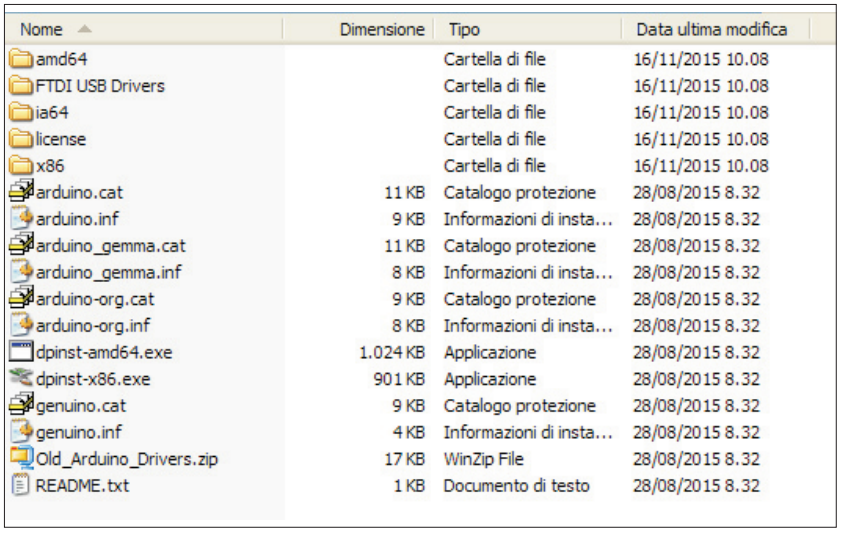
Dato che il connettore Tipo B non è più molto diffuso, non sottovalutate il fatto che potrebbe essere difficile trovare il cavetto necessario nel negozio d'informatica sotto casa. Dopo aver collegato il cavetto alla scheda, con il computer acceso potete collegare il cavetto al computer per attivare l'installazione del driver.
Con le versioni da Vista in poi l'installazione automatica va buon fine e si ottiene un dispositivo Genuino UNO nel gruppo "Porte COM ed LPT".

2011-2016 CC-BY-NC-ND



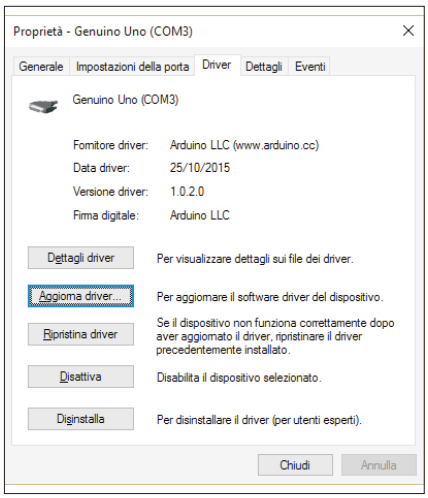


Se invece non è andata a buon fine e avete un "Dispositivo Sconosciuto", con il tasto destro cliccateci sopra e scegliete "Aggiorna Driver" per entrare nella procedura che vi permette di indicare al sistema operativo dove andare a cercare il driver giusto. Con la nostra scheda questo si trova nella cartella "Drivers" creata assieme alle altre all'interno della cartella principale dove è stato installato il software.



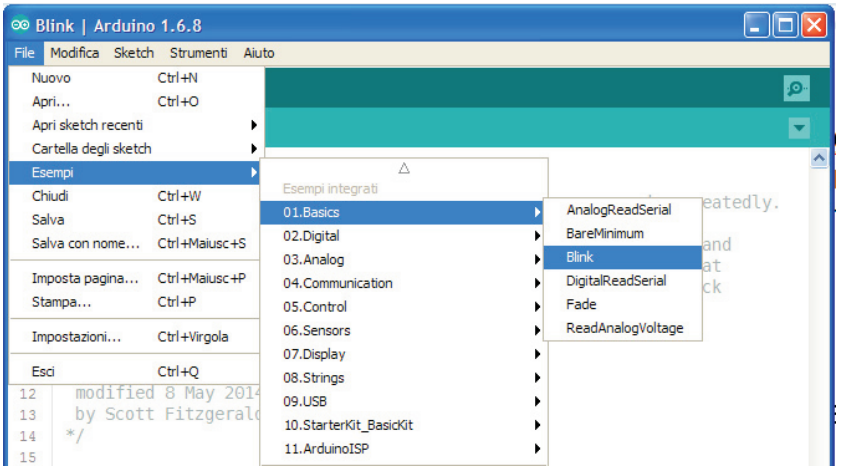
Nome	Dimensione	Tipo	Data ultima modifica
amd64		Cartella di file	16/11/2015 10.08
FTDI USB Drivers		Cartella di file	16/11/2015 10.08
ia64		Cartella di file	16/11/2015 10.08
license		Cartella di file	16/11/2015 10.08
x86		Cartella di file	16/11/2015 10.08
arduino.cat	11 KB	Catalogo protezione	28/08/2015 8.32
arduino.inf	9 KB	Informazioni di insta...	28/08/2015 8.32
arduino_gemma.cat	11 KB	Catalogo protezione	28/08/2015 8.32
arduino_gemma.inf	8 KB	Informazioni di insta...	28/08/2015 8.32
arduino-org.cat	9 KB	Catalogo protezione	28/08/2015 8.32
arduino-org.inf	8 KB	Informazioni di insta...	28/08/2015 8.32
dpinst-amd64.exe	1.024 KB	Applicazione	28/08/2015 8.32
dpinst-x86.exe	901 KB	Applicazione	28/08/2015 8.32
genuino.cat	9 KB	Catalogo protezione	28/08/2015 8.32
genuino.inf	4 KB	Informazioni di insta...	28/08/2015 8.32
Old_Arduino_Drivers.zip	17 KB	WinZip File	28/08/2015 8.32
README.txt	1 KB	Documento di testo	28/08/2015 8.32

Dato che l'IDE supporta ancora tutti i vari tipi di schede precedenti a UNO, sono presenti driver e cartelle per le schede basate su chip USB della FTDI, mentre UNO utilizza un ATmega16U2.
Date l'OK sulla scelta della cartella in cui cercare il driver e lasciate lavorare il sistema operativo che, dopo poco, scriverà che sta installando il driver. La procedura, a seconda delle caratteristiche della macchina e dei driver già installati potrà richiedere da meno di un minuto a qualche minuto. È importante lasciare la scheda collegata alla porta USB per non creare problemi al sistema di plug and play e comunque al termine del processo verrà mostrata la schermata di conferma dell'installazione.

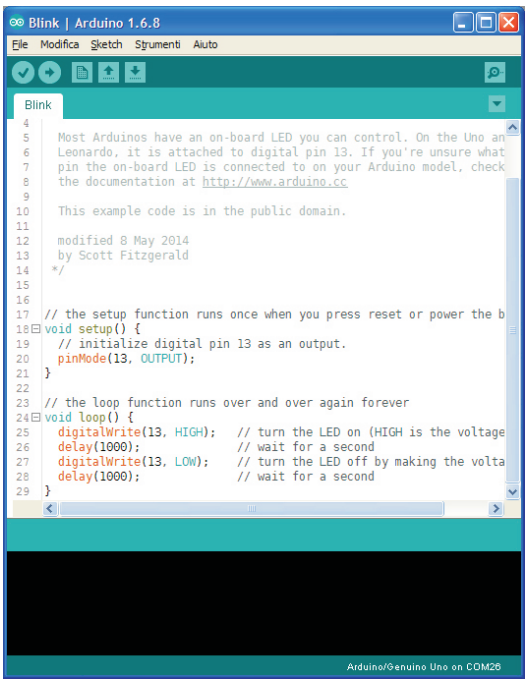


Da ora in poi, ogni volta che collegherete Arduino UNO al vostro computer, questo lo riconoscerà come una periferica di tipo seriale e lo posizionerà di conseguenza fra queste.
Nella nostra installazione, a Genuino UNO è assegnata la porta COM3, ma non è detto che sulla tua la situazione sia uguale. A seconda delle altre periferiche e degli altri driver presenti (ad esempio dell'hardware per il bluetooth), la prima porta COM disponibile potrebbe essere con una numerazione molto più alta. Questo non crea comunque alcun problema.
Per collaudare l'installazione puoi terminare la sessione del sistema operativo e riavviare la macchina senza la scheda collegata, quindi collegala e verifica se viene riconosciuta rapidamente e correttamente.

2011-2016 CC-BY-NC-ND



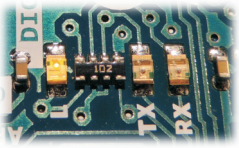
La seconda manovra di collaudo è un po' più articolata e vi farà caricare uno Sketch di esempio, compilandolo e scaricandolo sul vostro Genuino UNO.
Lanciate l'Arduino Software (IDE) e dal menu **File** scegliete **Esempi** -> **Basic** -> **Blink**.
Si aprirà una nuova finestra competa con il codice dello sketch caricato e "pronto all'uso".
Ora potete verificare l'installazione del software avviando la compilazione dello sketch con il pulsante di verifica (quello che sembra il tasto "V" in alto a sinistra). Nella barra azzurra in basso apparirà la scritta "Sto compilando" e dopo qualche istante nell'area della consolle (la finestra a fondo nero in basso) apparirà la scritta che conferma la creazione del codice oggetto pronto per essere caricato sulla scheda con i relativi dati di occupazione della memoria:




```
Lo sketch usa 1.066 byte (3%) dello spazio disponibile
per i programmi. Il massimo è 32.256 byte.
Le variabili globali usano 9 byte (0%) di memoria di-
namica, lasciando altri 2.039 byte liberi per le va-
riabili locali. Il massimo è 2.048 byte.
```

Il passo successivo è quello del caricamento del software sulla scheda utilizzando il pulsante Carica che è accanto a quello di Verifica.

Il processo di caricamento sulla scheda parte dopo qualche istante dal termine del processo di compilazione, con il trasferimento vero e proprio che viene evidenziato dal lampeggio dei led TX ed RX della scheda Arduino in varie ondate.
Al termine avrete come effetto il LED collegato al Pin 13 e contrassegnato dalla lettera **L** che lampeggerà lentamente con la cadenza 1 secondo acceso e 1 secondo spento. Potrebbe darsi che anche prima di caricare lo sketch blink il led fosse già lampeggiante e questo perché come collaudo dell'hardware viene utilizzato proprio questo programma che, quindi, potrebbe già essere attivo sulla scheda. Se qualcosa fosse andato storto, non



2011-2016 CC-BY-NC-ND

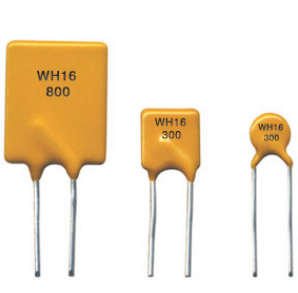
avresti visto lampeggiare i led RX e TX oppure l'Arduino Software (IDE) avrebbe visualizzato dei messaggi di errore.

Se ancora non vi fidate della vostra installazione e volete un'ulteriore prova, potete variare i valori delay(1000) portandoli a delay(100). Spostate il cursore sul primo valore, correggetelo e quindi fate lo stesso per il secondo, senza toccare null'altro. Premete il tasto **Carica** per verificare, compilare il codice e caricarlo in una sola operazione. Se avete fatto tutto giusto e l'installazione è OK vedrete il LED che lampeggia con cadenza di un decimo di secondo, quindi molto velocemente.

Gli altri ingredienti

Ora che gli elementi base per i nostri primi passi con Genuino UNO sono consolidati e funzionanti, possiamo dare uno sguardo a quello che vorremo utilizzare per fare qualche realizzazione pratica di un minimo spessore.
Come in una ricetta di cucina, oltre alle pentole e ai fornelli, dobbiamo mettere sul tavolo quello che poi combineremo e trasformeremo in un piatto finito. Per poterlo fare dobbiamo quindi acquisire un minimo di conoscenza sui componenti elettronici per riuscire sia a distinguerli, sia ad utilizzarli nel modo corretto.

Il componente più comune e meno "impegnativo" è la **resistenza**: un componente passivo che resiste al passaggio della corrente elettrica limitandola. Un po' come una strozzatura limita il fluire dell'acqua, la resistenza – in base al suo valore espresso in ohm – limita la corrente e ci aiuta a non bruciare ad esempio un LED alimentato a 5 volt. Le resistenze hanno il medesimo comportamento sia in corrente continua, sia in corrente alternata.
Le resistenze possono anche essere variabili tramite la rotazione di un perno (**potenziometri**) o in relazione a una grandezza fisica quale la luce (**fotoresistenza**) o la temperatura (**termistore**). Restano anche in questo caso componenti passivi, ma la loro struttura fisica gli consente di variare la resistenza in base alla dilatazione termica o alla conduttività di un semiconduttore fra due contatti.



2011-2016 CC-BY-NC-ND

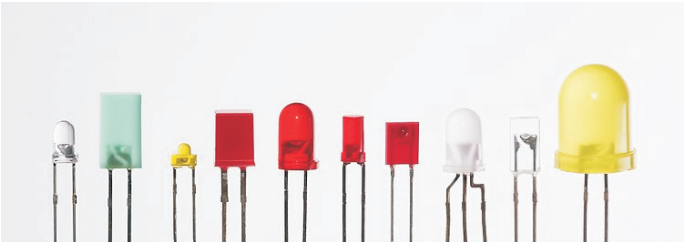
credit: wikipedia

Un **condensatore** è invece paragonabile a una membrana che separa due ambienti: l'aria non passa da una parte all'altra, mentre il suono che è vibrazione dell'aria, viene trasmesso da una parte all'altra. A seconda della rigidità della membrana e delle sue dimensioni, passano suoni bassi e acuti, o solo bassi o solo acuti e così via.
In elettronica, il condensatore isola dalla corrente continua, mentre lascia passare i segnali in alternata (come ad esempio l'audio da un lettore MP3 sull'uscita cuffia). Possono essere polarizzati o non polarizzati e svolgono anche un importante ruolo nella stabilizzazione e nella pulizia dell'alimentazione.

Ai **pulsanti** e agli **interruttori** spetta la funzione di aprire e chiudere i circuiti tramite un intervento manuale. I pulsanti possono essere normalmente aperti o chiusi, e lo stesso avviene per gli interruttori. I primi sono temporanei (si toglie la pressione e si ripristina lo stato iniziale), mentre i secondi mantengono lo stato in quanto hanno un meccanismo di scatto e ritenzione della posizione.
La stessa scheda Genuino UNO ha un pulsante per il reset, di tipo normalmente aperto, a corsa breve.

credit: wikipedia

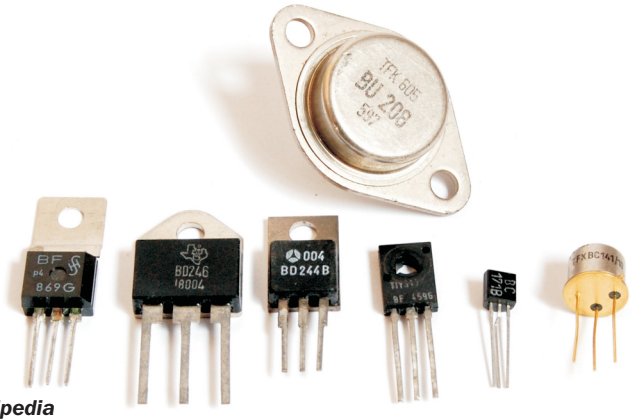
I **diodi** sono componenti attivi, a semiconduttore, che lasciano passare la corrente solo in una direzione, ovvero dal terminale positivo (**anodo**) a quello negativo (**catodo**) come anche indica il simbolo a freccia. Di diodi ne esistono diversi tipi e malgrado quelli tradizionali siano alla base di molti circuiti indispensabili nella vita quotidiana, quelli che godono di maggiore fama sono sicuramente i **LED**, o Light Emitting Diode. Questa tipologia di diodo ha avuto degli sviluppi notevoli negli ultimi anni per la sostituzione delle sorgenti di illuminazione:



2011-2016 CC-BY-NC-ND

sono molto più longevi dei filamenti delle lampadine e hanno una efficienza decisamente superiore. In pratica le lampadine LED durano di più e consumano di meno.

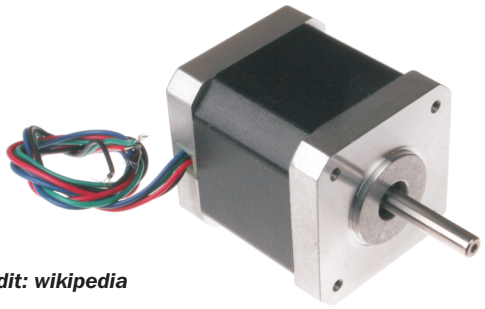
Fra gli altri tipi di diodo troviamo gli **Zener** che oltre a condurre nel normale senso negativo-positivo, entrano in conduzione quando viene applicata una tensione inversa superiore al voltaggio di Zener; sono molto utili per limitare e regolare la tensione in circuiti di alimentazione.



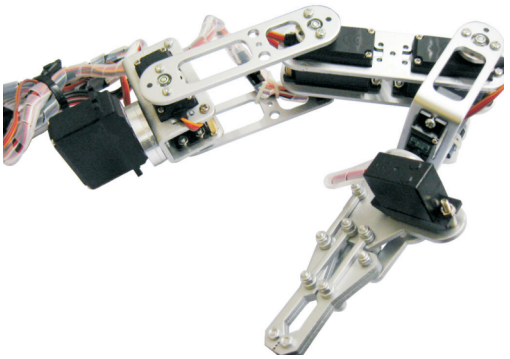
credit: wikipedia

Proseguendo nella carrellata troviamo i semiconduttori per eccellenza: i **transistor**. Questi dispositivi possono funzionare come interruttori allo stato solido o come amplificatori di segnale. Nel primo caso ci aiutano a gestire carichi importanti – come un motorino o una strip da qualche metro di LED – senza sovraccaricare le porte di Genuino, mentre nel secondo caso portano al livello richiesto un segnale basso, come ad esempio quello di un microfono.

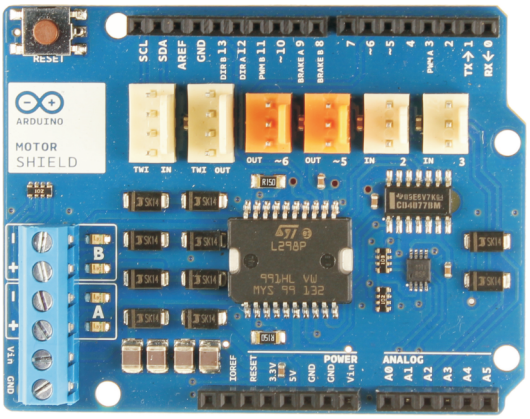
Chiudiamo questo veloce excursus sui vari componenti che potremo utilizzare con Genuino con i **motori**: quelli a spazzole in **corrente continua** e quelli **Servo**. I primi sono i classici motorini che alimentati in corrente continua girano con una velocità proporzionale alla tensione di alimentazione – ovviamente rispettando i limiti operativi -; i secondi nascono dal mondo dei modellini radiocomandati e sono fatti per avere un posizionamento preciso di un braccetto o di un ingranaggio che ruota su un arco di 180 o 270 gradi, ma non ha una rotazione continua e completa. I servo sono gestiti con un segnale digitale di tipo PWM – che Genu-



credit: wikipedia



2011-2016 CC-BY-NC-ND



ino genera senza problemi – e con una alimentazione a 5V: il duty cycle del PWM determina la posizione del braccetto. A livello esemplificativo, un motore a spazzole può far andare avanti e indietro un piccolo robot con delle ruote, mentre il servomotore può attivare un meccanismo di sterzo con estrema precisione o gestire un'articolazione, sempre con precisione. Se volete realizzare un progetto con motori in continua dovete obbligatoriamente aggiungere degli stadi di potenza e al proposito il **Motor Shield** ha proprio questi stadi pronti all'uso.

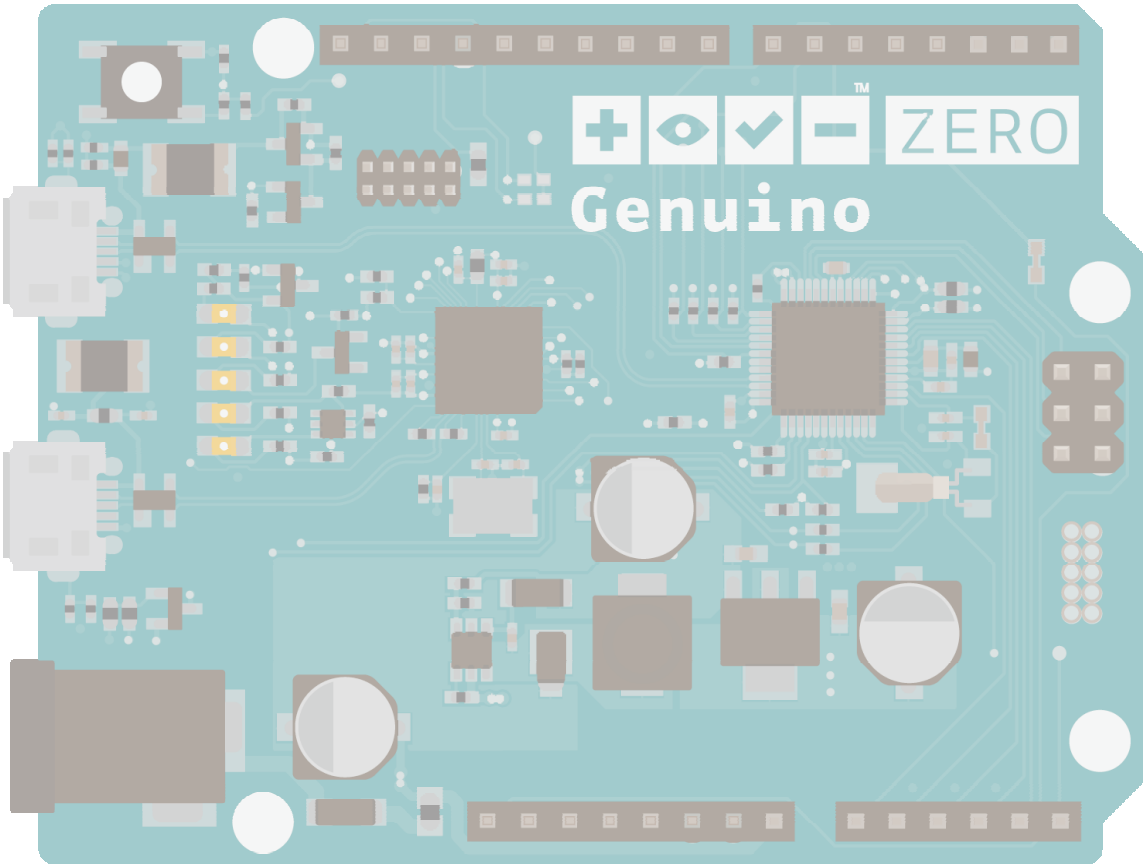
Per collegare velocemente fra loro Genuino e i componenti, la soluzione più rapida è la **breadboard**: una speciale basetta con dei contatti a molla su una matrice a righe e colonne. Bastano dei fili di collegamento e i componenti per realizzare in pochi istanti i circuiti, verificandone la funzionalità prima di creare un circuito su basetta millefori. I kit di sperimentazione prevedono spesso la presenza di una breadboard proprio per la sua praticità e per la possibilità di lasciare i reofori dei componenti come nuovi. Di modelli di breadboard ne esistono diversi e si differenziano per il numero di fori e per la presenza delle linee di alimentazione ai due lati delle file di contatti. La breadboard da 400 fori è il modello tipico per chi vuole fare un po' di sperimentazione con le schede della famiglia Genuino e dei componenti elettronici quali pulsanti e LED. Se invece si pensa di utilizzare anche schermi a cristalli liquidi, lo spazio disponibile potrebbe non bastare e la versione lunga il doppio risulta la scelta più adatta. Assieme alle breadboard, non vanno dimenticati i fili di collegamento che possono essere sostanzialmente di due tipi: spezzoni di filo piegato e

Colori dei LED e resistenze

Quando si usano i LED è importante ricordare che non vanno alimentati superando la loro corrente massima di lavoro, utilizzando sempre una resistenza in serie del valore opportuno. Il valore della resistenza dipende sia dalla tensione di alimentazione, sia dalla corrente che vogliamo far fluire nel diodo, sia dalla sua caduta di tensione (o tensione di lavoro). Giusto per rendere le cose più interessanti, la tensione a cui il LED lavora dipende dal suo colore, passando da 1,8V dei rossi fino ai 3,5V dei bianchi. Quindi, in pratica, quando utilizzate con Arduino dei LED, la resistenza per limitare l'assorbimento va scelta in base al colore del LED stesso. Per semplificarti la vita, ecco una pratica tabella in cui trovi il colore del LED e la resistenza per limitare a 10 o 20 mA la corrente. Dato che Arduino ha dei limiti sul massimo della corrente che può essere erogata complessivamente, circa 150 mA, se vuoi collegare molti led su molte porte, può essere utile fornire a ciascuno solo 10 mA.

Colore	Volt caduta	20mA @ 5V	10mA @ 5V
Infrarosso	1.5	180 ohm	390 ohm
Rosso	2.0	150 ohm	330 ohm
Arancio	2.0	150 ohm	330 ohm
Giallo	2.1	150 ohm	330 ohm
Verde giallo	2.2	150 ohm	330 ohm
Verde smeraldo	3.3	100 ohm	180 ohm
Blu	3.3	100 ohm	180 ohm
Bianco	3.3	100 ohm	180 ohm
UV	3.3	100 ohm	180 ohm
Blu viola (406 nm)	4.6	22 ohm	47 ohm

2011-2016 CC-BY-NC-ND



credit: Arduino.cc

2011-2016 CC-BY-NC-ND

La nostra prima realizzazione pratica: contiamo fino a tre

Se fossimo sui fornelli il primo piatto sarebbe un uovo all'occhio di bue, mentre in elettronica si comincia accendendo le luci. Noi accenderemo tre LED in sequenza

Il primo progetto pratico richiede sempre più impegno rispetto ai successivi in quanto devi prendere confidenza con tutte le sfumature dell'ambiente di lavoro sia dal punto di vista software, sia da quello hardware. Pur nella sua semplicità, Genuino ha alcuni aspetti da tener presente per non incappare in piccoli scivoloni e intoppi imprevisti.

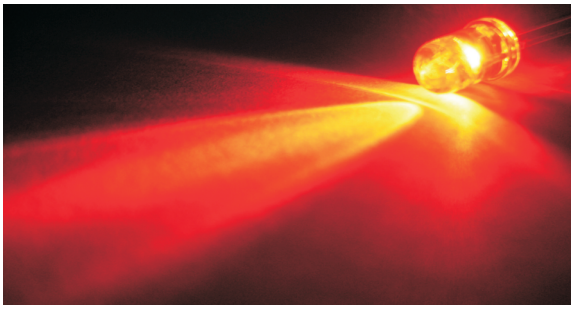
Il primo progetto

Accendere in sequenza tre LED rossi, ripetendo all'infinito la sequenza non è certo rivoluzionario, ma ci permette di partire dall'inizio e arrivare in fondo con i rischi di manovre errate ridotti al minimo e con la possibilità di far evolvere il nostro "giocattolo" con poco sforzo.

Gli obiettivi dell'esercitazione sono i seguenti:

- individuare le caratteristiche dei componenti usati
- prendere confidenza con la scheda Genuino UNO e i suoi PIN
- acquisire la manualità per collegare i componenti sulla breadboard
- imparare a descrivere il processo in un diagramma di flusso
- imparare le basi della sintassi del linguaggio di programmazione
- trasformare i blocchi del diagramma di flusso in codice
- svolgere il processo di scrittura, compilazione e upload di un vostro sketch.

I componenti



I LED rossi da utilizzare in questo progetto sono di tipo standard, da 5mm e con il corpo colorato o trasparente. Potreste averli già in casa o esserveli procurati in un kit di base.
Ciascun LED va collegato a Genuino tramite una resistenza che è stata calcolata per un assorbimento di 10 mA, con 5V di alimentazione e 2 volt di caduta di tensione del LED (valore che deriva dal colore del LED, che a sua volta deriva

2011-2016 CC-BY-NC-ND

dal tipo di semiconduttore impiegato). La formula per calcolare il valore è:

resistenza = (volt alimentazione - caduta tensione LED)/corrente LED

ovvero:

(5.0 V - 2.0 V) / 0,01 A = 300 ohm

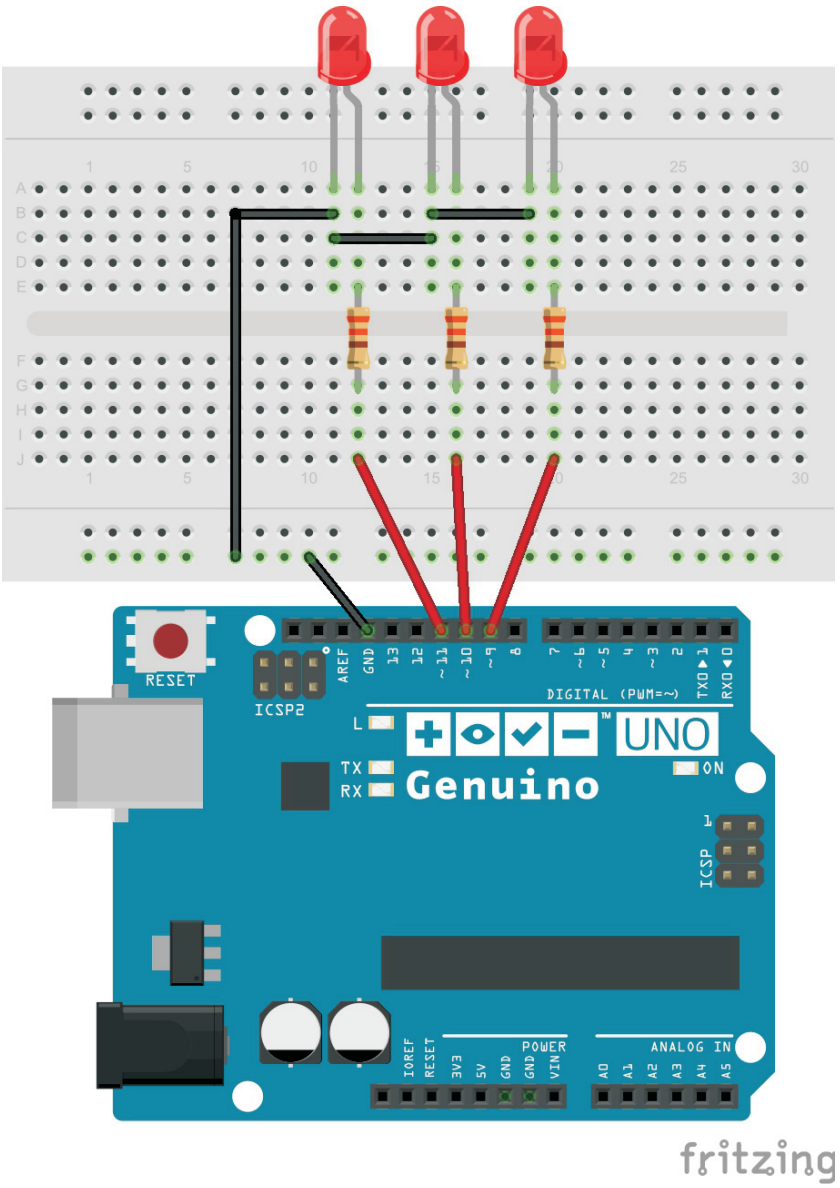
Il collegamento della breadboard alla scheda Genuino si può fare con i ponticelli rigidi o i cavetti flesssibili, in foto qui sotto.

Poiché non ci sono resistenze da 300 ohm, si usa il valore disponibile immediatamente successivo, che è 330 ohm. Nel capitolo precedente abbiamo riportato in un box le resistenze da utilizzare con tutti i vari colori di LED, con 10 e 20 milliampere di assorbimento. Con la formula riportata qui sopra siete invece autonomi nelle vostre decisioni, da prendere anche in base alle caratteristiche di potenza del LED utilizzato. Ricordate comunque che Genuino NON va fatto erogare o assorbire più di 20 milliampere su qualsiasi porta digitale, mentre se si arriva a 40 mA si rischia concretamente il suo danneggiamento.

Per mettere insieme i tre LED e le tre resistenze, vi servirete di una **breadboard**, ovvero una basetta in plastica con all'interno una serie di contatti disposti in orizzontale e verticale. Le due file di contatti in alto e in basso sono da utilizzare per portare l'alimentazione. Abbiamo evidenziato come sono collegate con le righe rosse e blu. In mezzo alla basetta si trovano invece delle colonne, separate centralmente, che hanno cinque fori ciascuna. Il loro collegamento è evidenziato con le righe grigie.

Il passo - ovvero la spaziatura fra i fori - è quello dei circuiti integrati DIL (Dual In Line), lo stesso del chip ATMega328 usato sulla scheda Genuino e delle due file di connettori ai bordi della scheda stessa.
Sulla breadboard i componenti si infilano nei fori senza bisogno di saldature grazie ai reofori, ma dato che le molle dei contatti sono fatte per durare, gli spezzoni di filo di rame tendono a piegarsi e rompersi. Meglio utilizzare quelli fatti apposta per le breadboard - con degli appositi terminali - o mettere mano a un saldatore per dare una stagnatura di rinforzo.

2011-2016 CC-BY-NC-ND



Circuito elettrico e schema

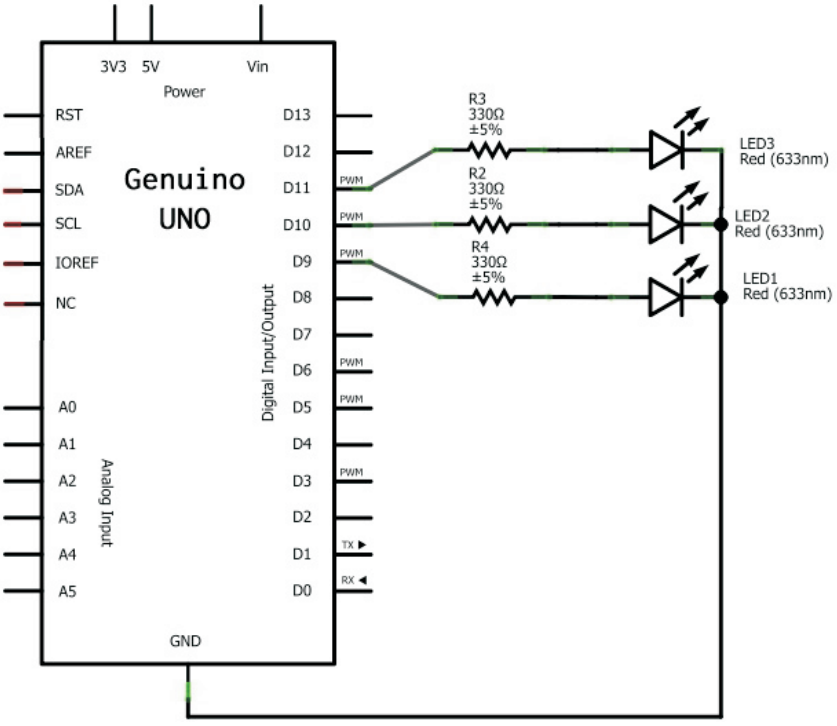
Per il nostro circuito hai bisogno di quattro spezzoni di filo che porteranno i segnali e l'alimentazione dalla scheda di Genuino UNO alla breadboard. I collegamenti sono schematizzati nel disegno qui accanto. Lo schema è sotto il disegno ed entrambi sono realizzati con Fritzing che può essere scaricato gratuitamente da **fritzing.org**.

In questo schema dovete notare due cose: il positivo dei LED è collegato a Genuino attraverso le tre resistenze e le porte utilizzate sono D9, D10 e D11.

Perché i LED si accendano, ciascuna porta deve presentare una tensione positiva e questo corrisponde al livello logico 1 o "HIGH".

Per far funzionare il circuito al contrario, potete girare i LED con il negativo verso Genuino e spostare il collegamento comune da massa (GND o 0V) a 5V. In questo modo per accendere i LED ciascuna porta deve essere a massa, ovvero assumere lo stato logico 0 o "LOW".

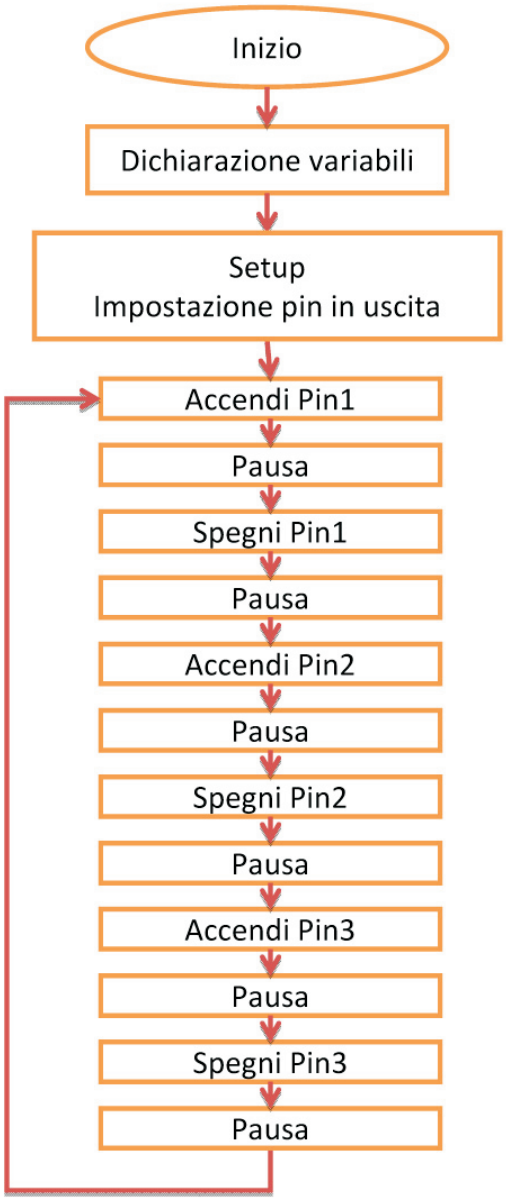
Ciascuna porta digitale può quindi assumere il ruolo di massa o positivo per il componente a cui è collegata e questo spiega il perché della limitazione della corrente sia in erogazione (quando la porta è a 5V), sia in assorbimento (quando la porta è a 0V).



Il diagramma di flusso

Come avete visto nel capitolo sui diagrammi di flusso, ora è arrivato il momento di descrivere quello che volete fare come una sequenza di azioni e di situazioni, con un riferimento alle funzio-

2011-2016 CC-BY-NC-ND



nalità di Genuino, ma senza scendere nei dettagli tecnici.
A livello di prima approssimazione, dobbiamo dichiarare le variabili che useremo, attivare le tre porte digitali in uscita con il setup e continuare ad accendere i LED in sequenza, restando in questo loop all'infinito.
NOTA: per ragioni didattiche abbiamo scelto di gestire l'accensione dei LED in sequenza; avremmo potuto utilizzare altre soluzioni più compatte che usano il valore di un ciclo come elemento da sommare a un valore base per definire dinamicamente il numero del pin da attivare. Quando avrai portato a termine l'esercizio così come lo abbiamo descritto, prova tu a trovare delle altre strade per accendere i LED in sequenza in modo più compatto.

Le regole di base e i primi accorgimenti

Prima di passare alla scrittura del codice vero e proprio nell'Arduino Software (IDE), entriamo nel dettaglio di quello che dovremo scrivere e di come scriverlo.
Innanzi tutto con il C utilizzato per programmare Genuino UNO, *le lettere maiuscole sono diverse dalle minuscole* e quindi bisogna fare attenzione anche a questo aspetto. Un altro elemento fondamentale è l'utilizzo delle parentesi graffe "**{}**", difficili da digitare sulle tastiere italiane, ma indispensabili per racchiudere il codice relativo a tutte le strutture di controllo.
Con i punti e virgola "**;**" che terminano ogni riga di un blocco di codice si conclude il campionario di peculiarità della grammatica dell'Arduinese. Non mettere il "**;**" dove serve causa sicuramente qualche problema di compilazione e lo stesso accade se manca la chiusura di qualche parentesi - sia tonda, sia graffa.
Durante la digitazione, l'editor dello sketch cerca di darvi una mano evidenziando in colore tutte le parole riservate a comandi per le funzioni, le strutture e le parole che definiscono uno stato o un tipo di dato; i colori sono quelli che abbiamo riportato nei titoli a pagina 32 e 33. Se digitando un comando questo resta nero, allora avete sbagliato a scriverlo. Per le parentesi, quando vi posizionate in uno sketch dopo una parentesi, l'editor evidenzia la corrispondente parentesi di apertura o chiusura riquadrandola con un bordo sottile colorato.

```
1 void setup() {
2   pinMode (redPin,  OUTPUT);  // imposta i pin come uscita
3   pinMode (greenPin,OUTPUT);
4   pinMode (bluePin, OUTPUT);
5 }
```

Potete riscontrare quanto descritto nel pezzo di schermata dell'IDE con un blocco di codice: vediamo pinMo-

2011-2016 CC-BY-NC-ND

de in rossiccio, con la M maiuscola, seguito da una parentesi in cui si trova la variabile in nero e lo stato OUTPUT in blu. Nella riga iniziale si trova anche il cursore "|" appena dopo la parentesi graffa aperta ({)che definisce la struttura void setup() in fondo a questa si trova la parentesi graffa di chiusura (}), evidenziata dal rettangolo proprio perché il cursore è accanto a quella di apertura.
Nel blocco successivo abbiamo invece inserito apposta degli errori e la compilazione non va a buon fine.

```
5 void setup() {
6   pinMode (redPin,  OUTPUT);  // imposta i pin come uscita
7   pinmode (greenPin,Outut);
8   PinMode (bluePin, output);
9 }
```

In questo esempio, sbagliando le maiuscole e le minuscole, "pinmode", "PinMode", "OutPut" e "output" sono passati da parole riservate (riconosciute e colorate dall'editor) a variabili che, non essendo state dichiarate in precedenza, creano un errore di "variabile non definita". L'evidenziazione in rosa è fatta dal compilatore che segnala la prima occorrenza di un errore nel codice.
È interessante notare come si possano dare alle variabili dei nomi apparentemente uguali alle parole riservate, semplicemente giocando sulle maiuscole e minuscole. Vi sconsigliamo di farlo come abitudine, per non rendere il vostro codice di difficile lettura.

Le funzioni del nostro programma

Passiamo ora alle strutture di controllo e ai comandi che useremo in questo primo sketch.

int variabile;
Per non scrivere il numero del pin nelle funzioni che lo riguardano, definiamo tante variabili intere quanti sono i pin che useremo. "int" va scritto interamente minuscolo e va seguito dal testo che useremo per la variabile seguito dal ";" di chiusura della riga. Se dopo la stringa di testo mettiamo "=" e un valore avremo sia dichiarato la variabile, sia assegnato ad essa un valore. Tornando al nostro programma, useremo questa seconda modalità per dichiarare i pin e la Pausa.

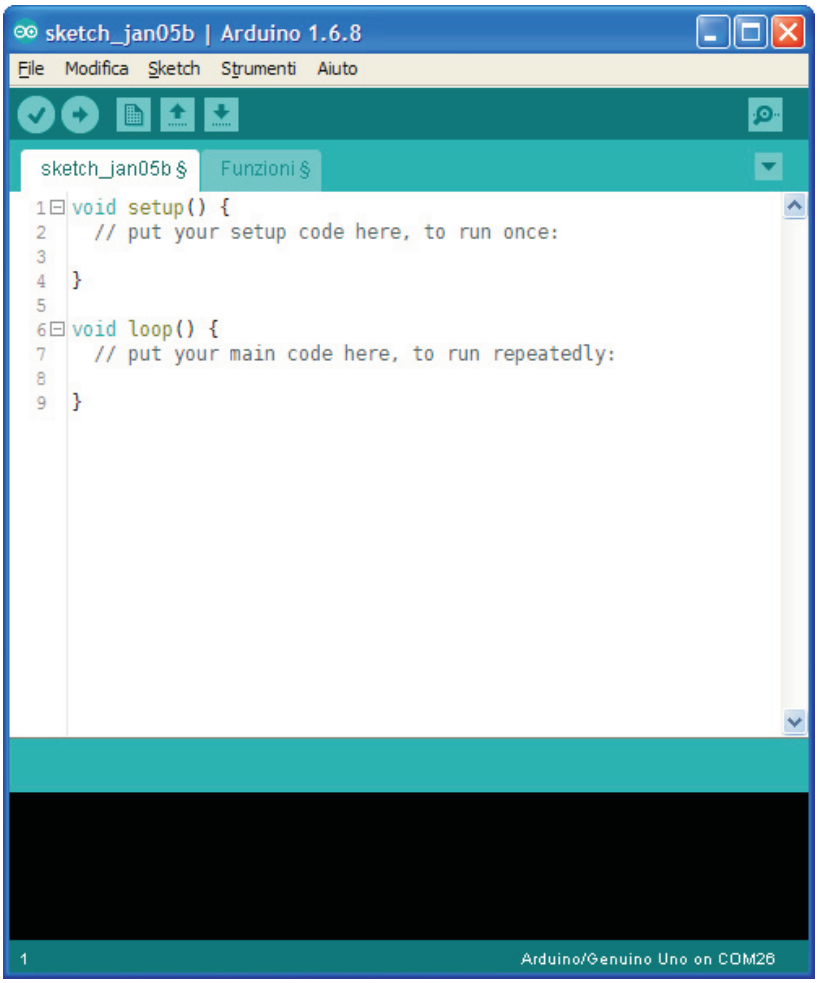
pinMode(pin, modo);
Prima di poter utilizzare i pin digitali, va definito per ciascuno di essi se sono nel modo INPUT o OUTPUT. Se non li definiamo con l'istruzione pinMode, questi restano in uno stato ad alta impedenza che non crea disturbo agli eventuali circuiti collegati.

delay(millisecondi);
Quando c'è bisogno di aspettare prima di passare alla riga successiva di codice, si

 2011-2016 CC-BY-NC-ND

usa l'istruzione delay seguita dal valore dell'attesa in millisecondi. Il valore può essere rappresentato da un numero, una variabile o una espressione. Questa istruzione è da un lato molto comoda, ma dall'altro tiene Genuino impegnato a non fare nulla, sordo a qualsiasi stimolo, fino al termine dell'attesa. Per questo motivo si tratta di una funzione definita come "bloccante". Nella programmazione si cerca di usare soluzioni diverse che non blocchino l'esecuzione ed è anche probabile che prima o poi la cosa sia risolta a livello di linguaggio. Se volete approfondire c'è uno sketch chiamato "Blink without delay" negli esempi "Digital" che esemplifica una delle possibili soluzioni.

digitalWrite (pin, livello)
Con questa istruzione possiamo definire il livello logico di ciascuna porta digitale. Il pin può essere una variabile, mentre il livello può essere LOW o HIGH (parole riservate) oppure una variabile di tipo bool che assume quindi i valori 0 e 1.



Scriviamo lo sketch

Lanciamo l'Arduino Software (IDE) facendo doppio click sull'icona di Arduino e attendiamo che venga caricata l'interfaccia di programmazione. Avendo eseguito l'installazione come descritto un paio di capitoli, il sistema è già pronto e configurato; nella riga in basso la scritta "Arduino/Genuino Uno on COMxx" con xx corrispondente al numero della porta di comunicazione creata dal driver vi conferma che è tutto pronto anche per la programmazione della scheda.

Ogni volta che si lancia il software, viene proposto un documento con la struttura minima e un nome composto da "sketch_" seguito da "mese giorno" e da una lettera dell'alfabeto. In questo modo il software permette di salvare senza preoccuparsi di dare un nome, pur creando un archivio di file che ha un proprio senso logico e non sovrascrive accidentalmente i lavori precedenti.

2011-2016 CC-BY-NC-ND

Il numero in basso a sinistra indica la riga a cui si trova il cursore "|", mentre l'area nera appena sopra è la finestra in cui appaiono i messaggi durante la compilazione e il caricamento sulla scheda.
Il pulsante in alto a destra con il triangolino verso il basso, sotto la lente di ingrandimento, ci permette di gestire i tab dello sketch, aggiungendone di nuovi o gestendo quelli esistenti. Uno sketch salvato mantiene la struttura dei tab e quindi è possibile lavorare su più listati - in tab indipendenti - gestendoli come unico file da aprire e salvare. I tab sono utili per mettere da parte dei pezzi di codice da usare in modo ripetitivo con copia e incolla, oppure come area temporanea o ancora come parcheggio per caratteri speciali come ad esempio le parentesi graffe.

VERIFICA

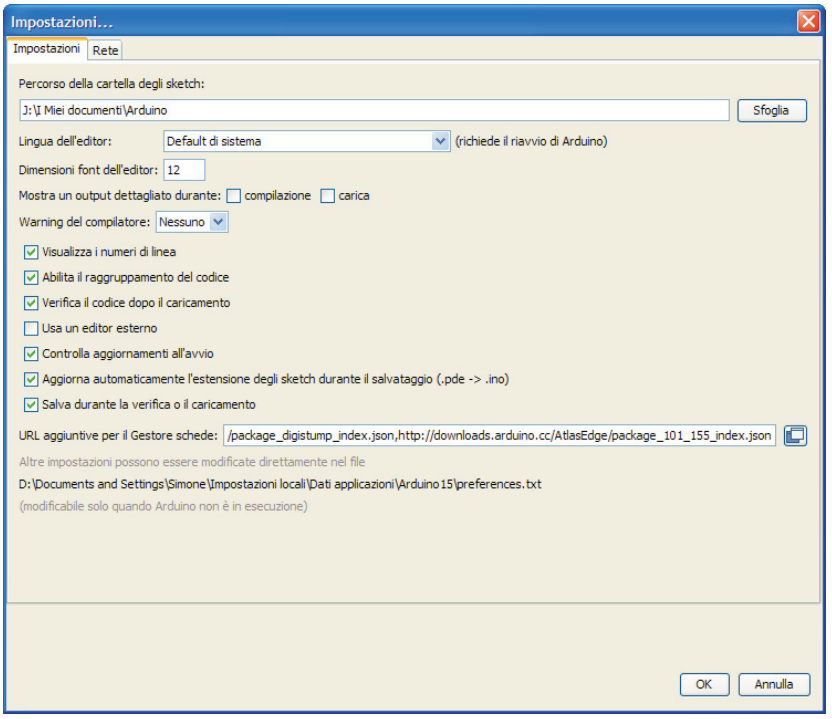
CARICA

NUOVO

APRI

SALVA

Se carichiamo uno sketch da disco, questo si apre in una nuova finestra con tutti i suoi eventuali tab. Per salvare il codice via via che viene scritto, si usa l'icona "salva" che è la più a destra nella serie di cinque, in alto a sinistra. L'interfaccia segnala anche quando il file è stato modificato - e quindi va salvato - mettendo il simbolo "§" accanto al nome nel tab.



Nelle nostre catture avrete notato che si vedono a sinistra i numeri di linea: una vecchia abitudine che ci portiamo dietro da quando abbiamo iniziato a programmare più di 35 anni fa e che per noi ha una grande utilità. Se anche voi volete avere questa funzionalità, potete aprire la scheda delle impostazioni

2011-2016 CC-BY-NC-ND

(File -> Impostazioni...) e attivare l'opzione "Visualizza i numeri di linea". Approfittiamo di questa indicazione anche per segnalarvi che dalle impostazioni potete cambiare la dimensione del carattere usato per il testo, utile se state usando un computer con uno schermo ad elevato dpi (con pixel fitti e molto piccoli, per capirci) e attivare anche altre funzionalità. Se ad esempio volete capire cosa vi state risparmiando a livello di comandi e complicazioni per passare dal vostro sketch a del codice eseguibile, attivate "Mostra un output dettagliato durante: compilazione": alla prossima verifica nella parte in basso vedrete scorrere una serie di scritte che sono legate proprio al processo di compilazione. Prima dell'Arduino Software (IDE) molti di quei comandi andavano impartiti a mano.

Ora che avete i riferimenti necessari per usare l'interfaccia, potete procedere con la scrittura del codice così come lo riportiamo qui di seguito.
Per posizionare i commenti - preceduti da "//" vi consigliamo di usare il tasto di tabulazione.
Per creare le parentesi graffe aperta "{" e chiusa "}" ci sono vari metodi: dal copia e incolla predisponendo un tab con le sole due parentesi, alla combinazione sul tastierino numerico ALT+0123 e ALT+0125.
Aprendo un nuovo documento avrete già le due strutture base e le parentesi graffe al loro posto.

```
1  int  Pin1 = 9;      // Inizializziamo le variabili
2  int  Pin2 = 10;     // assegnando i tre pin alle porte digitali da 9 a 11
3  int  Pin3 = 11;
4  int  Pausa = 1000;
5
6  void setup() {      // una sola volta, all'avvio, definiamo le modalità
7    pinMode (Pin1, OUTPUT);    // tutti i pin in modalità output
8    pinMode (Pin2, OUTPUT);
9    pinMode (Pin3, OUTPUT);
10 }
11
12 void loop() {       // da qui tutto si ripete all'infinito
13   digitalWrite (Pin1, HIGH);    // attiviamo il primo pin
14   delay (Pausa);                // pausa da 1000 millisecondi
15   digitalWrite (Pin1, LOW);     // disattiviamo il primo pin
16   delay (Pausa);                // altra pausa
17   digitalWrite (Pin2, HIGH);    // ora con il secondo pin
18   delay (Pausa);
19   digitalWrite (Pin2, LOW);
20   delay (Pausa);
21   digitalWrite (Pin3, HIGH);    // e infine con il terzo pin
22   delay (Pausa);
23   digitalWrite (Pin3, LOW);
24   delay (Pausa);
25 }
```

Il codice da inserire è riportato nel riquadro in questa pagina; digitatelo con cura.
Fate attenzione ai colori delle varie parole, per avere conferma che non ci siano stati errori di digitazione. Anche i ";" in fondo alle righe possono essere stati erroneamente inseriti come ":", ma in questo caso il compilatore vi avviserà con degli errori.

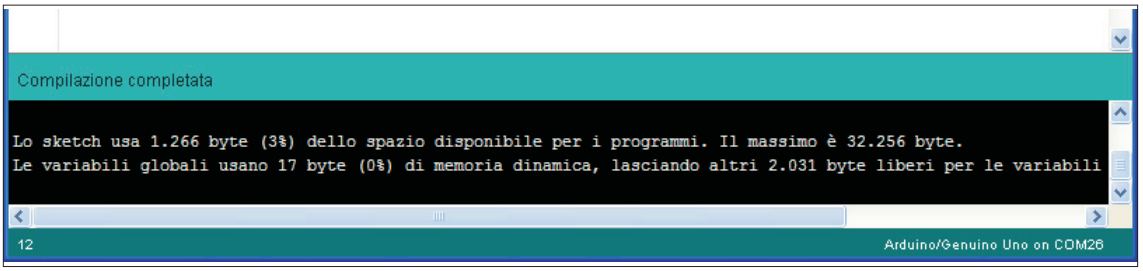
Ora più che mai è il caso di salvare il codice digitato dando al progetto un

2011-2016 CC-BY-NC-ND

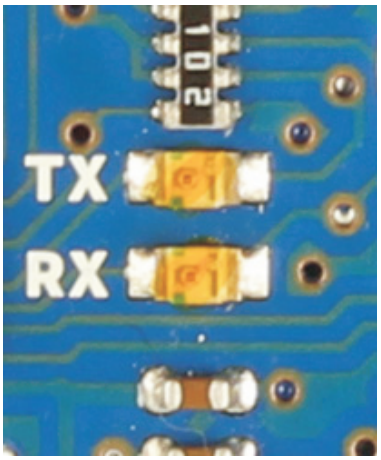
nome. Noi abbiamo scelto Cap6_New per indicare che è lo sketch del sesto capitolo della nuova versione del libro, ma ovviamente ogni nome va bene.
Quando l'IDE salva uno sketch, crea una cartella con il nome scelto e all'interno crea i file con estensione ".ino". Tutte le versioni precedenti salvavano con l'estensione ".pde", ancora comunque riconosciuta dal nuovo IDE 1.0, ma ora sconsigliata.

Verifica e Compilazione

Questa procedura, avviata con un click sull'icona "verifica" in alto a sinistra, scatena una serie di azioni che trasformano il vostro programma in codice eseguibile dalla scheda Genuino UNO. Premetela e seguite il processo grazie alla barra di avanzamento presente in basso a destra. Se c'è qualche errore di digitazione, il compilatore interromperà il processo di verifica e vi segnalerà quello che secondo lui non è corretto. Una banda rosa evidenzierà la riga "incriminata", ma non è detto che il problema sia proprio lì. Solitamente i problemi derivano dall'errata scrittura di una parola riservata o dalla mancanza di un punto e virgola. Anche le parentesi sono potenziali cause d'errore.



Se invece non avete fatto errori, il processo di verifica si concluderà con una riga che indica la dimensione del codice oggetto e ricorda la memoria totale disponibile.



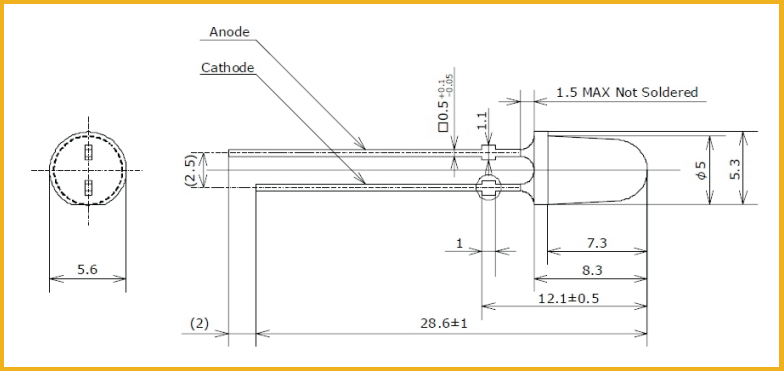
Upload

E siete finalmente giunti al caricamento del primo programma che avete seguito e scritto sin dall'inizio. Controllate che tutto l'hardware sia ancora collegato correttamente sia al computer - tramite USB - sia ai tre LED e alla breadboard. In caso affermativo, premete sul secondo pulsante in alto a sinistra e attendete che il vostro sketch sia prima compilato e poi trasferito alla scheda. Potete vedere il momento preciso in cui il codice viene caricato osservando il lampeggìo dei due LED TX e RX sulla scheda: è in questo momento che entra in

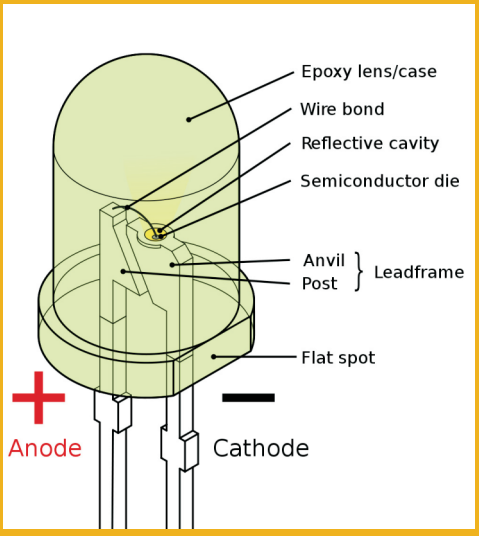
2011-2016 CC-BY-NC-ND

Anatomia del LED

I LED sono componenti attivi e polarizzati, ovvero sono basati su semiconduttore e hanno come gli altri diodi un catodo e un anodo. Le forme e le dimensioni dei LED sono state standardizzate; i più comuni hanno un diametro di 3 o 5 mm e sono con una lente in materiale plastico trasparente ottenuta dando una forma specifica all'involucro.
I due reofori non hanno la medesima lunghezza e l'anodo (+) è quello più lungo. Il catodo (-) è invece contrassegnato da una tacca sul colletto dell'involucro.



Un'altra peculiarità dei led in questo formato è la sezione quadrata dei reofori, con un piccolo ma marcato ingrossamento a pochi millimetri dall'involucro. Questo aiuta nel montaggio a distanziare sempre nel medesimo modo i LED dallo stampato. Se il LED ha l'involucro trasparente, si può osservare come il chip semiconduttore sia posizionato in una specie di vaschetta ottenuta sagomando opportunamente il metallo. Il chip emette luce sia dalla parte superiore, sia da tutti i bordi e quindi la vaschetta ha la funzione di un riflettore. L'angolo di divergenza della luce prodotta dal LED dipende dalla forma dell'involucro che, con la sua curvatura, è una vera e propria lente che mantiene più o meno focalizzata la luce. Continuando ad osservare l'interno, si nota un filo che collega l'anodo al chip nel riflettore.Continuando ad osservare l'interno, si nota un filo che collega l'anodo al chip nel riflettore.



gioco il bootloader e avviene il trasferimento del codice.

Serve il debug?

Al termine nella finestra apparirà la conferma del corretto upload e dopo pochi istanti vedrete i tre LED accendersi in sequenza. Se si accendono solo due LED o un LED o tutti e tre, ma poco, è necessario avviare il processo di debug (caccia all'errore) per verificare se è sbagliato il cablaggio o se c'è un problema nel software.
Se non si accende nulla, controllate se ci sono messaggi di errore a fine upload. Controllate anche che Genuino UNO dia segni di vita sia collegandolo all'alimentazione tramite USB, sia durante il processo di upload con il lampeggìo dei suoi LED.
Per verificare se è l'hardware o il software ad avere problemi, controllate l'orientamento del LED sulla breadboard. Se avete uno o due LED che si accendono, scambiate il loro filo verso la scheda Genuino con quello che non va e verificate se ancora il LED non funziona: in questo caso avete un problema hardware in quanto siete certi che Genuino lo sta accendendo. Se i LED che sulle altre porte vanno, collegati alla porta che non accende il LED, non si accendono a loro volta, avete un problema nel software e dovete controllare lo sketch.
Questo approccio in cui partite da un elemento certo e funzionante (la porta che va o il LED che si accende) vi consente di individuare rapidamente la natura del problema.

2011-2016 CC-BY-NC-ND

Basta poco per migliorare le cose

Aggiungiamo un potenziometro per variare la velocità dello scorrimento

Nel progetto precedente abbiamo fatto accendere tre LED in sequenza, con pause identiche sia per il tempo di accensione, sia di spegnimento. Se avete osservato con attenzione il listato, avrete notato che abbiamo utilizzato la variabile "Pausa" per definire la durata della pausa, applicata con una serie di istruzioni di ritardo delay (Pausa) messe a ogni cambio di stato dei LED. Con questa struttura software, gestire la velocità di accensione dei LED diventa molto semplice: agendo sulla variabile Pausa, modifichiamo tutti i valori di delay () in un colpo solo.

Dall'analogico al digitale

Genuino UNO dispone di ben **sei ingressi analogici** ai quali possiamo collegare qualsiasi dispositivo che presenti una tensione – su un'impedenza elevata – compresa fra 0 e 5V rispetto alla massa della scheda. Il fatto che l'impedenza sia elevata determina che non venga praticamente influenzato con un assorbimento di corrente significativo il dispositivo che genera la tensione.
Attenzione però, questa caratteristica va tenuta presente se la tensione da misurare dipende proprio dalla presenza di un assorbimento di corrente, come ad esempio con una fotoresistenza.
Genuino UNO elabora e utilizza solo bit e byte, quindi la tensione applicata agli ingressi analogici viene convertita in valori digitali attraverso un processo che viene definito come conversione **Analogico Digitale** (A/D). Il chip ATmega328P dispone internamente di tutta la circuiteria analogica e digitale necessaria al processo, purché sia nell'ambito di misurazione previsto.
L'intervallo gestito, da 0 a 5V, viene suddiviso in tanti livelli discreti quanti sono i valori digitali gestiti; più sono i valori e maggiore sarà la precisione del valore rappresentato digitalmente in quanto ciascun gradino diventa più piccolo e rappresenta un intervallo più ristretto. Nel nostro caso, i valori che Genuino riesce a rappresentare sono su 10 bit e quindi vanno da 0 a 1023. Se i bit fossero solo 8, i valori sarebbero da 0 a 255, ma rappresenterebbero sempre un intervallo da 0 a 5V, solo che ogni gradino sarebbe più grande. Trovate una trattazione più dettagliata di questo tema nel capitolo 4, nel sottocapitolo dedicato ai convertitori A/D.
Mettendo un po' di numeri bianco su nero, ecco la situazione:

Range misurato:	**0 – 5 volt**
Risoluzione:	**10 bit – 1024 valori**
Analog Reference:	**sì (possibilità di usare una soglia diversa da 5V)**
Precisione su 0/5V:	**+/- 0,00488 volt**

Per progetti specifici, sono disponibili chip di conversione da analogico a digitale con caratteristiche specifiche quali numero di bit e velocità di acquisizione.

2011-2016 CC-BY-NC-ND

Se dovessimo misurare un valore più ristretto, per esempio da 0 a 3,7V, potremmo fornire 3,7V al pin **AREF** ed avere una lettura limitata a quell'intervallo. Perché questo succeda dobbiamo anche utilizzare la funzione **analogReference(EXTERNAL)**. Tornando al nostro progetto, il dispositivo ideale per applicare sul pin analogico di Genuino una tensione variabile è il **potenziometro**, ovvero una resistenza che ha tre terminali: due presentano il valore nominale del potenziometro, mentre il terzo dipendente dalla posizione del cursore.

Qui sotto un potenziometro rotativo e il suo schema interno di funzionamento.

I potenziometri

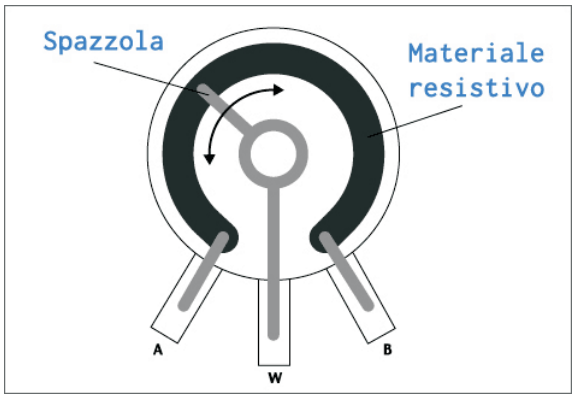
I potenziometri sono un componente passivo di grande utilità e solo con la massiccia diffusione dei dispositivi controllati digitalmente se ne sta un po' perdendo l'uso.

Tutti i dispositivi audio Hi-Fi avevano, una volta, nel potenziometro un componente basilare per regolare il volume, i toni, il bilanciamento e altro ancora; non c'era "Volume" audio senza potenziometro neppure nelle radio e nelle televisioni; se un parametro funzionale era "regolabile", solitamente c'era un potenziometro per farlo.

Oggi è quasi tutto regolato con pulsanti "+" e "-" o "<" e ">", anche perché fra le comodità irrinunciabili ci sono i telecomandi che obbligatoriamente devono utilizzare un sistema di controllo alternativo al buon vecchio potenziometro.

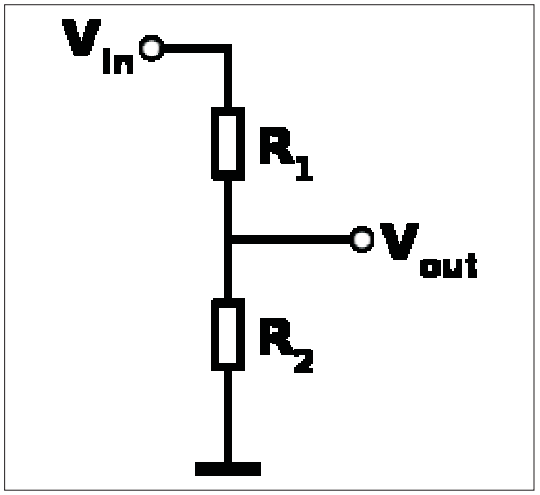
Lo stesso è avvenuto nei dispositivi portatili, dove è sparita la rotella del volume, sostituita da pulsanti digitali che pilotano microcontrollori audio dedicati.

Prima che diventi un componente di nicchia, vale la pena capire come sia realizzato. Per funzionare, il potenziometro utilizza uno strato di materiale resistivo (come ad esempio della polvere di carbone) steso su un arco di circonferenza (o un segmento lineare per gli slider) ai cui estremi sono posti due terminali. La resistenza fra questi due terminali è il valore nominale del potenziometro. Fissato ad un perno (o a un cursore) si trova il contatto strisciante che, scorrendo in un punto variabile della pista resistiva, realizza un circuito elettrico con una resistenza il cui valore è compreso fra zero e il valore nominale a seconda della posizione. Da notare che il potenziometro offre sempre due valori di resistenza, ottenuti fra il centrale (cursore) e ciascuno dei due terminali. La somma dei due valori è, salvo piccole tolleranze, il valore nominale del potenziometro.



2011-2016 CC-BY-NC-ND

Le possibilità di collegamento di un potenziometro sono di fatto due: con i due terminali collegati a due punti del circuito e il cursore che spostandosi realizza un partitore di tensione variabile, oppure con un solo terminale e il cursore per avere una semplice resistenza di valore variabile.



Qui a sinistra il partitore di tensione con due resistenze.

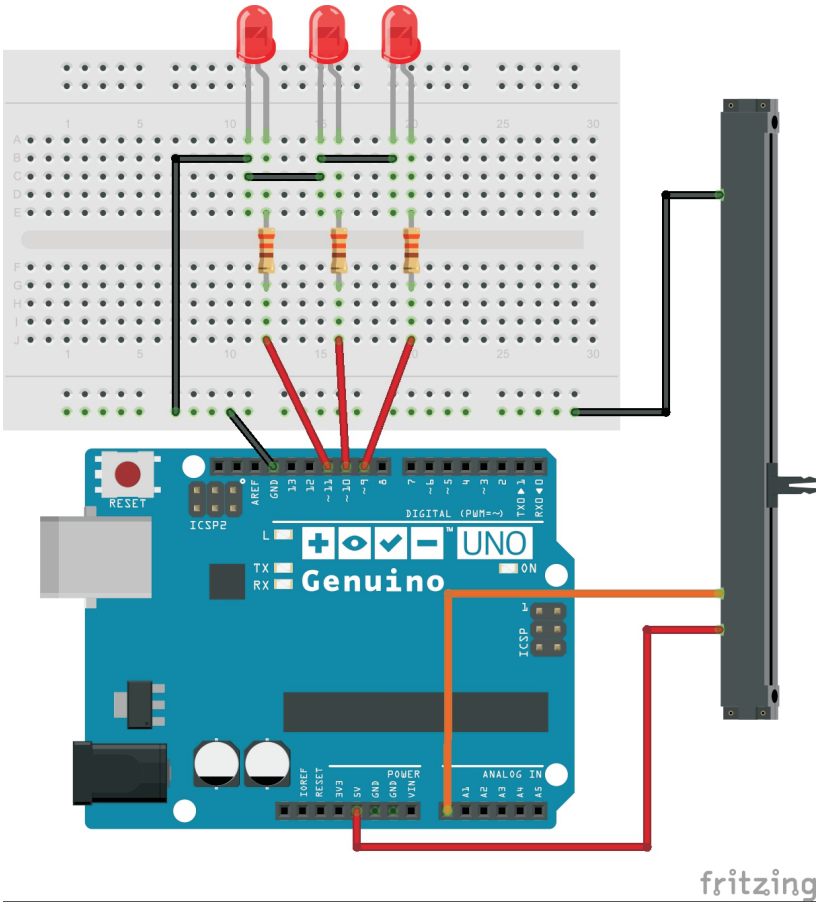
Vout = Vin * R2/(R1+R2)

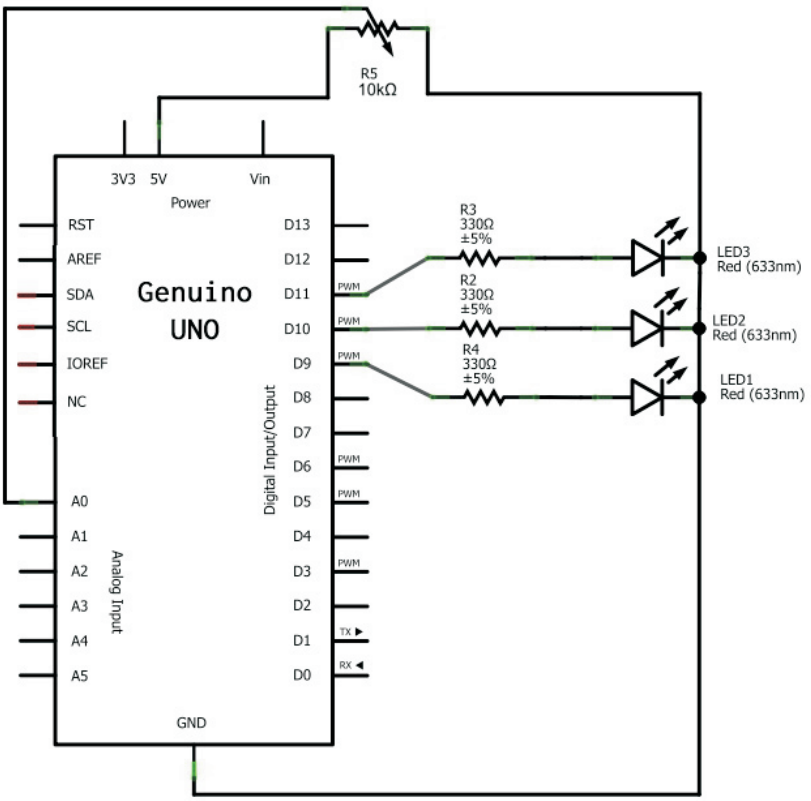
La formula vi indica che il rapporto fra la somma delle resistenze e il valore di R2 determina quanta tensione si trovi al capo di R2. Questo però riguarda solo il tema della tensione, mentre la corrente che fluisce è soggetta alla legge di Kirchhoff. Fortunatamente per noi, con Genuino UNO l'impedenza del piedino di ingresso analogico è alta e quindi la corrente assorbita risulta trascurabile e non comporta quindi un impatto significativo sulla tensione presente sull'uscita del circuito.
Allo stesso modo, se utilizzassimo il potenziometro come pura resistenza variabile, con il secondo terminale lasciato libero o collegato al cursore, ci troveremmo con una lettura della tensione con una piccolissima variazione a causa dell'impedenza elevata dell'ingresso. Questa infatti sarebbe R2 nel nostro partitore e quindi con 10k ohm di potenziometro R1 – valore che vi consigliamo per il progetto – e circa 100K di impedenza di ingresso, avremmo una variazione fra 5 e 4,55 volt. Pensare di usare un potenziometro di valore più alto servirebbe a poco in quanto per portare a valori vicini a 0V l'ingresso, la resistenza R1 dovrebbe avere valori enormi. Il medesimo ragionamento va fatto per tutti i dispositivi a resistenza variabile che vorremo collegare a un ingresso analogico: collegarli fra il positivo e l'ingresso, senza una resistenza dall'ingresso verso massa per creare un partitore di tensione, ci impedirebbe di avere una variazione di tensione.

Lo schema elettrico

Alla luce di tutte le varie considerazioni, possiamo quindi aggiungere al circuito già realizzato per lo scorso progetto il nostro potenziometro da 4700 ohm. Serviranno tre spezzoni di filo per collegare i tre terminali del potenziometro alla breadboard, mentre dalla scheda di Arduino UNO alla breadboard porteremo la linea del positi-

 2011-2016 CC-BY-NC-ND





vo e quella dalla porta A0 degli ingressi analogici.

Aggiorniamo il diagramma di flusso

Rispetto al progetto precedente, ora dobbiamo leggere il valore presente sull'ingresso A0 e assegnarlo alla variabile Pausa, già presente anche nello sketch del capitolo 6. Il pin di ingresso analogico non necessita di alcuna inizializzazione e quindi non abbiamo interventi da fare né alla parte di dichiarazioni di variabili, né alla parte di setup. La posizione del blocco in cui leggiamo il potenziometro è all'inizio del loop principale, così da assegnare alla Pausa un valore reale sin dalla prima iterazione.

Poiché la lettura avviene solo all'inizio di ogni ciclo completo, il nostro programma cambia la velocità di rotazione dei LED solo dopo un tempo pari al valore della pausa moltiplicato per sei. Se Siamo con la pausa massima – 1023 millisecondi – anche ruotando il potenziometro a metà o al minimo vedremo un cambio di velocità dopo 6 secondi. Con il potenziometro a metà corsa, la lettura avviene dopo 3 secondi, mentre con il potenziometro al minimo l'intero ciclo prende qualche millisecondo.

La funzione per leggere il pin analogico

La funzione **analogRead(pin)** può essere utilizzata senza dover definire in fase di setup alcuna modalità di funzionamento. I pin sono infatti in sola lettura e non è possibile trasformarli in output analogici: la loro elevata impedenza non influenza significativamente i circuiti eventualmente collegati rendendone inutile una modalità di "disattivazione" come

2011-2016 CC-BY-NC-ND

invece accade per i pin digitali. Ti ricordiamo che l'uscita analogica è simulata dai pin digitali tramite la modalità PWM.
Questa funzione restituisce un valore fra 0 e 1023, ovvero un valore a **10 bit**, che può essere assegnato a una variabile, oppure utilizzato direttamente in un'altra funzione.

Il nuovo listato

Partendo da quanto già fatto nello scorso capitolo, creiamo un nuovo sketch semplicemente scegliendo **File -> Save As... -> Capitolo_7**. Avremo così una nuova copia del programma, pronto per la modifica all'altezza delle righe 14, 15 e 16.

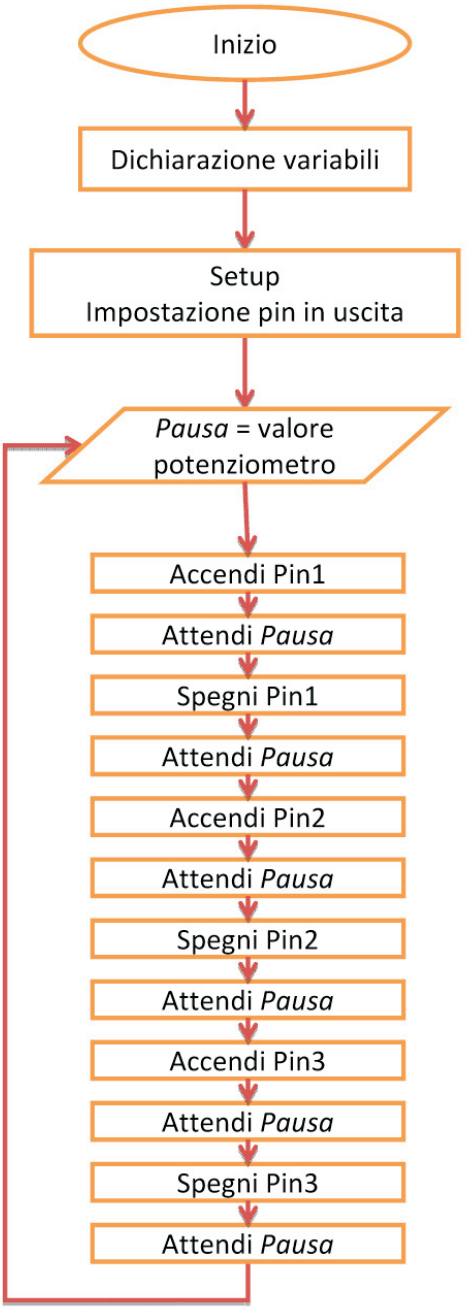
La situazione originale è:

```
…
void loop() {          // da qui tutto si ripete all'infinito
  digitalWrite (Pin1, HIGH);   // attiviamo il primo pin
  delay (Pausa);               // pausa da 1000 millisecondi
  digitalWrite (Pin1, LOW);    // disattiviamo il primo pin
…
```

e deve diventare

```
…
void loop() {  // da qui tutto si ripete all'infinito

  Pausa = analogRead (0);     // Leggiamo il potenziometro
                              // e assegniamo a PAUSA il valore
  digitalWrite (Pin1, HIGH);  // attiviamo il primo pin
  delay (Pausa);              // pausa da 1000 millisecondi
  digitalWrite (Pin1, LOW);   // disattiviamo il primo pin
…
```

Praticamente basta inserire la riga dove assegniamo a Pausa il valore letto da analogRead() sul pin analogico 0.



Qui sopra il nuovo diagramma di flusso, realizzato aggiungendo la gestione del potenziometro e della pausa al diagramma di flusso del capitolo precedente.

Salvataggio e test

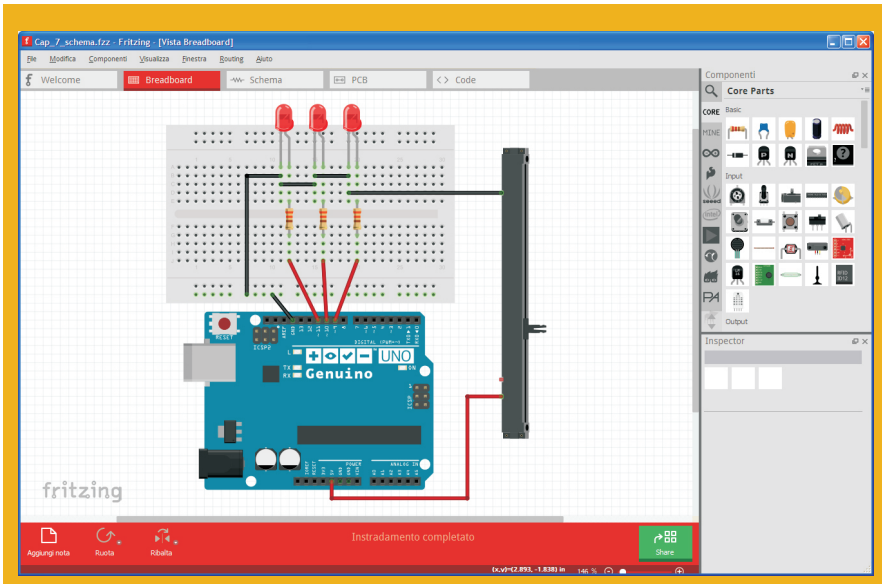
Ora potete salvare il nuovo sketch usando l'apposita icona e proseguire con la fase di verifica e compilazione che si innesca con l'icona a sinistra. Se non ci sono errori – cosa molto probabile – potete proseguire con il caricamento sulla scheda, che avrete nel frattempo collegato al computer nella nuova configurazione hardware con il potenziometro, come indicato dal disegno con la breadboard.

 2011-2016 CC-BY-NC-ND

```
int  Pin1 = 9;      // Inizializziamo le variabili
int  Pin2 = 10;     // assegnando i tre pin alle porte digitali da 9 a 11
int  Pin3 = 11;
int  Pausa = 1000;

void setup() {      // una sola volta, all'avvio, definiamo le modalità
  pinMode (Pin1, OUTPUT);     // tutti i pin in modalità output
  pinMode (Pin2, OUTPUT);
  pinMode (Pin3, OUTPUT);
}

void loop() {  // da qui tutto si ripete all'infinito

  Pausa = analogRead (0);        // Leggiamo il potenziometro
                                 // e assegniamo a PAUSA il valore
  digitalWrite (Pin1, HIGH);     // attiviamo il primo pin
  delay (Pausa);                 // pausa da 1000 millisecondi
  digitalWrite (Pin1, LOW);      // disattiviamo il primo pin
  delay (Pausa);                 // altra pausa
  digitalWrite (Pin2, HIGH);     // ora con il secondo pin
  delay (Pausa);
  digitalWrite (Pin2, LOW);
  delay (Pausa);
  digitalWrite (Pin3, HIGH);     // e infine con il terzo pin
  delay (Pausa);
  digitalWrite (Pin3, LOW);
  delay (Pausa);
}
```

Via alle danze

Appena l'IDE avrà terminato di trasferire il software sulla scheda, vedrete i tre LED accendersi in sequenza, con una velocità che dipende immediatamente dalla posizione del cursore del potenziometro. Se l'avrete messo circa a metà, vedrete un ritmo da circa un secondo per ogni LED, mentre se lo spostate a tre quarti da un lato e dall'altro il ritmo diventa sensibilmente più veloce o più lento.



I nostri schemi pratici

Lo splendido programma che utilizziamo per realizzare gli schemi pratici ed elettrici - **Fritzing** - è nato nel 2007 presso la University of Applied Science Potsdam e dal 2010 è gestito dalla fondazione senza scopo di lucro *Friends of Fritzing*. La Fondazione si impegna a mantenere il progetto aggiornato e open-source, raccogliendo donazioni e pianificando gli sviluppi. Se siete interessati a contribuire a questo progetto, potete farlo **qui**.

2011-2016 CC-BY-NC-ND

Adesso tocca a voi

Questo progetto si presta molto bene alla sperimentazione sul codice in quanto inserisce un controllo esterno molto semplice per variare un parametro. Le idee per migliorare e modificare il comportamento dei tre LED nella loro sequenza di accensione possono essere parecchie e per prendere maggiore confidenza con Genuino vi consigliamo di dedicare un po' di tempo proprio su questo sketch.

Fra le cose che vi suggeriamo ci sono:

1. La modifica della pausa su una scala temporale diversa
2. La trasformazione della somma delle pause "acceso" e "spento" in un tempo costante
3. E per i più intraprendenti, l'accensione di uno solo dei tre LED in base alla posizione del cursore del potenziometro.

Se proprio non avete idea di come fare, potete leggere il prossimo paragrafo, altrimenti dedicatevi immediatamente alla sperimentazione.

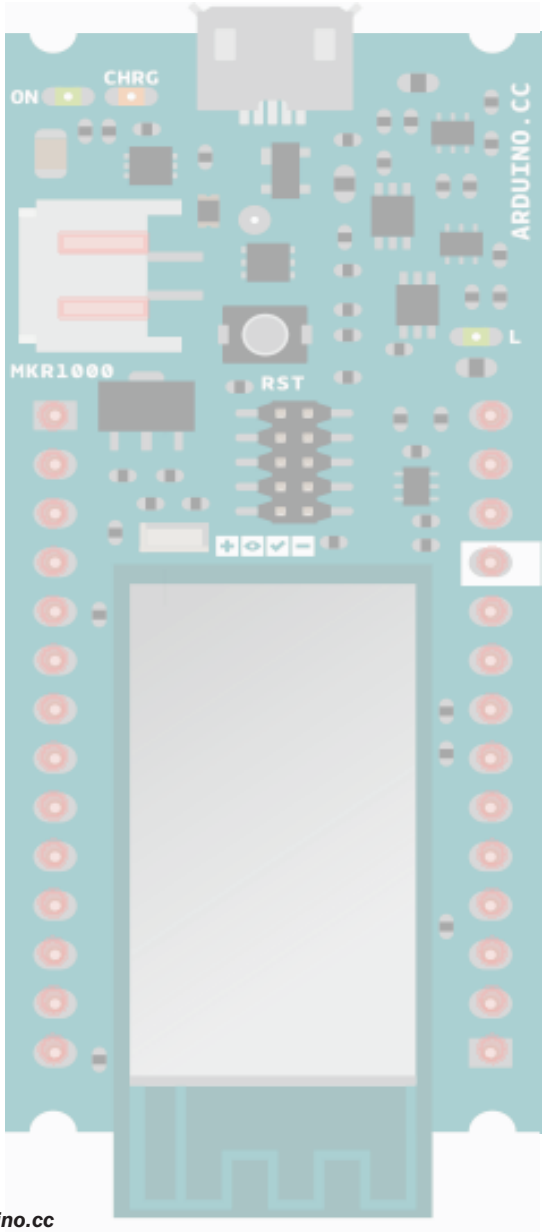
Spoiler…

1. Potete dividere o moltiplicare il valore letto dalla funzione analogRead() per definire un nuovo intervallo in cui Pausa può variare. Ricordate che l'espressione matematica può essere tutta su una sola riga e che potete anche definire un valore minino della Pausa sommando un valore costante. Per passare da un intervallo a un altro – da quello letto in analogico a quello della pausa - avete anche a disposizione una funzione specifica: map(value, fromLow, fromHigh, toLow, toHigh).

2. Per ottenere questo risultato dovete rendere **simmetriche** le due pause fra acceso e spento, lasciando alla prima il valore Pausa, e alla seconda il valore massimo dell'intervallo MENO il valore di Pausa. Il totale sarà pari al valore massimo della Pausa, in ogni situazione, mentre il rapporto fra acceso e spento varierà in base alla posizione del potenziometro. Se dividete il valore letto per 8, avrete che ogni LED si accende e spegne in circa 128 millisecondi e come effetto una variazione di luminosità percepita.

3. In questo caso dovete aggiungere un controllo per accendere i LED, dove se il valore è compreso fra 0 e 340 accendete il primo, se è fra 341 e 681 accendete il secondo e se è fra 682 e 1023 accendete il terzo. Alternativamente potete dividere il valore letto per 340 e avrete una variazione fra 0 e 2, da usare poi per accendere il LED corrispondente con la funzione digitalWrite(pin,stato); in questo caso ricordate che il primo pin è il numero 9.

2011-2016 CC-BY-NC-ND



credit: Arduino.cc

2011-2016 CC-BY-NC-ND

Dopo le luci animate, ora creiamo dei suoni

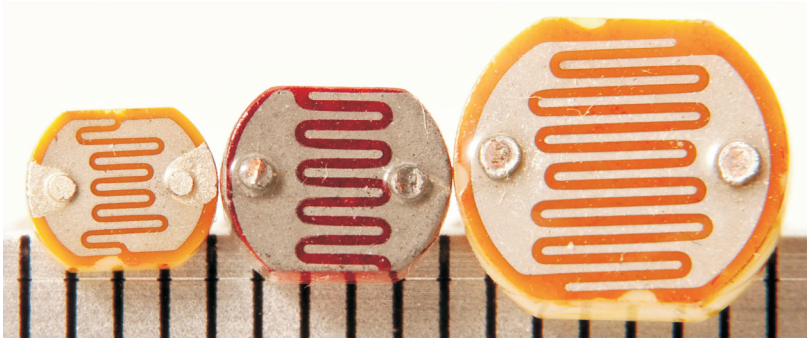
Modifichiamo qualche elemento e cambiamo radicalmente il risultato: un generatore di suoni pilotato dalla luce ambientale

Fin ad ora abbiamo imparato a usare la funzione per leggere l'ingresso analogico e acquisire un dato numerico; abbiamo appreso come tramite un partitore di tensione sia possibile far leggere a Genuino una variazione di una resistenza; abbiamo anche iniziato a utilizzare le variabili per gestire la durata di una pausa. Siamo quindi pronti per cambiare completamente tipo di progetto pur utilizzando le medesime conoscenze.

La fotoresistenza

Un componente che ha sempre colpito la fantasia degli sperimentatori è la **fotoresistenza**: un componente passivo che varia il proprio valore da circa un megohm al buio, fino a poche migliaia di ohm in piena luce. La sostanza chimica che permette alla fotoresistenza di comportarsi in questo modo è solitamente il **solfuro di cadmio** – in inglese Cadmium Sulfide - e per questo i nomi che trovi per questo componente sono vari: CdS cell, Light Dependant Resistor (LDR) e Photoresistor.

I modelli esistenti sono vari, ma la loro anatomia di base è comune: ogni contatto porta a una struttura conduttiva a pettine che ha di fronte quella dell'altro contatto, senza che i singoli denti – intercalati fra loro - si tocchino.



Qui sopra tre diverse fotoresistenze il cui valore dipende dal semiconduttore e dalla lunghezza della serpentina.

Fra i due "pettini" si trova il solfuro di cadmio. Se guardiamo invece la striscia di materiale sensibile alla luce come elemento di riferimento, questo ha la forma di una serpentina fra due contatti metallici.

Se raddrizzassimo la serpentina, otterremmo una striscia molto lunga con i contatti da ciascun lato e in mezzo la sottile striscia di solfuro di cadmio. In base alla lunghezza della striscia e alla distanza fra i due contatti, si ottiene un comportamento diverso dal punto di vista elettrico. Questo significa che ogni modello ha una resistenza massima e una minima specifiche.

Un'altra caratteristica delle fotoresistenze è la loro **velocità di risposta**: le variazioni vengono riflesse da un cambio di resistenza che inizia a stabilizzarsi dopo circa un decimo di secondo e se la variazione è veloce, la fotoresistenza può non essere in grado di seguirla. In caso di passaggio da luce a buio, molto rapidamente ci sarà un aumento di resistenza, ma solo dopo circa 10 secondi la fotoresistenza si sarà stabilizzata sul suo valore più elevato.

I dati tecnici delle fotoresistenze utilizzano il **LUX** come unità di misura della luce che colpisce il sensore. L'utilizzo di una unità di misura che esprime un flusso lumi-

2011-2016 CC-BY-NC-ND

La scheda tecnica di una fotoresistenza.

ELECTRO-OPTICAL CHARACTERISTICS RATING (TA)= 23°C UNLESS OTHERWISE NOTED

SYMBOL	CHARACTERISTIC	TEST CONDITIONS	MIN	TYP	MAX	UNITS
R_D	Dark Resistance	After 10 sec. @ 10 Lux @ 2856 °K	0.5			MΩ
R_l	Illuminated Resistance	10 Lux @ 2856 °K	9		20	KΩ
S	Sensitivity	LOG(R100)-LOG(R10)** / LOG(E100)-LOG(E10)***		0.6		Ω/Lux
λrange	Spectral Application Range	Flooded	400		700	nm
λpeak	Spectral Application Range	Flooded		570		nm
t_r	Rise Time	10 Lux @ 2856 °K		60		ms
T_f	Fall Time	After 10 Lux @ 2856 °K		25		ms

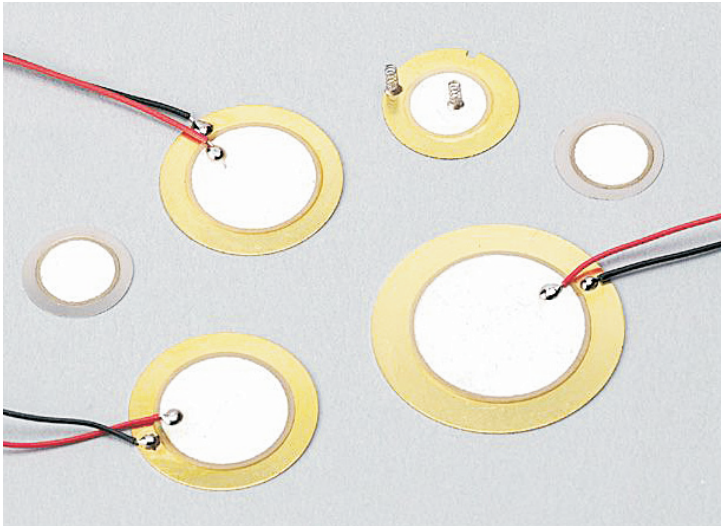
**R100, R10: cell resistances at 100 Lux and 10 Lux at 2856 °K respectively .
***E100, E10: luminances at 100 Lux and 10 Lux 2856 °K respectively.

noso è abbastanza intuibile in quanto il materiale della fotoresistenza varia il proprio valore in base alla quantità di fotoni che lo colpisce.
Nel box sul lux abbiamo riportato un po' di esempi di valori rispetto alla situazione ambientale.
Sulle specifiche, il valore indicato come resistenza più bassa è solitamente riferito ad un flusso luminoso di 10 lux, quindi con illuminazione più forte si ottengono valori ancora più bassi. Anche la lunghezza d'onda a cui la fotoresistenza è sensibile fa parte delle caratteristiche: nel caso del solfuro di cadmio si va da 400 a 700 nanometri, che corrispondono allo spettro dal rosso al blu, con l picco nella zona del verde giallo. Gli infrarossi e gli ultravioletti non influiscono sulla fotoresistenza. Questa sensibilità ricalca da vicino la curva di sensibilità dell'occhio umano.
Nella **tabella qui sopra**, le caratteristiche di una fotoresistenza comune.

Il trasduttore acustico

I trasduttori piezoelettrici sono solitamente composti da un disco di ottone su cui è depositato lo strato attivo, a sua volta metallizzato in superficie creando così il secondo contatto elettrico.

I dispositivi che trasformano un segnale elettrico in una variazione di pressione dell'aria – ovvero un'onda acustica – sono chiamati **trasduttori acustici**. Gli altoparlanti dei diffusori audio e delle cuffiette dei lettori MP3 sono i trasduttori acustici più tradizionali e conosciuti, ma esistono degli altri trasduttori molto diffusi che - non essendo visibili - non sono altrettanto famosi. Se avete un orologio digitale al polso e questo dispone di un allarme sonoro per la sveglia, allora contiene al suo interno un trasduttore piezoelettrico; il vostro forno a microonde emette un suono quando il conto alla rovescia arriva a zero e se è una nota fissa, anche questa è prodotta da un cicalino basato su **trasduttore piezoelettrico**.
Gli altoparlanti si basano su un magnete fisso e una serie di spire solidali a un cono sospeso e capace di oscillare per produrre dei suoni. La corrente che attraversa le spire genera in base alla sua intensità un campo magnetico - perpendicolare alle spire – che a sua volta interagisce con quello del magnete fisso e sposta il cono replicando le variazioni del segnale. L'intervallo di frequenze gestito dall'altoparlante dipende dalle caratteristiche fisiche dell'altoparlante stesso e per questo si trovano due o tre altoparlanti diversi in ciascun diffusore acustico di qualità: uno per gli acuti, uno per i medi e uno per i bassi. Acuti e medi possono essere gestiti da un unico altoparlante nei sistemi più compatti.



Come abbiamo indicato, l'intensità del campo magnetico dipende dalla corrente che attraversa le spire secondo le formule dell'elettroma-

2011-2016 CC-BY-NC-ND

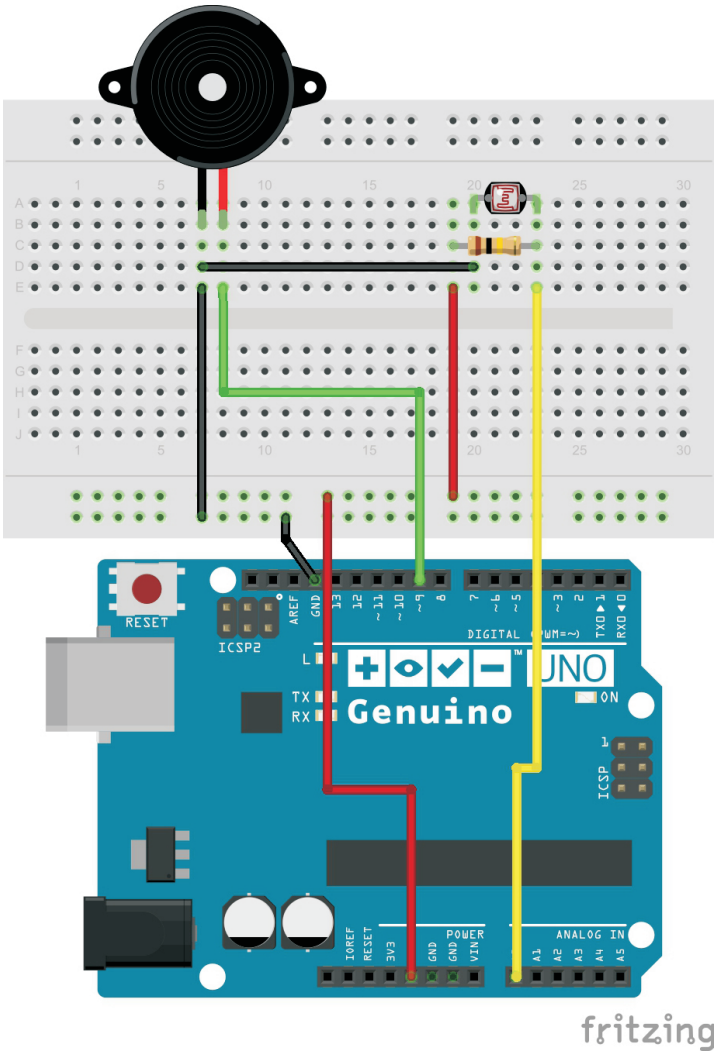
gnetismo e quindi gli altoparlanti tendono ad essere dei dispositivi "di potenza", ovvero da gestire con circuiti di amplificazione. La resistenza della spira è con valori compresi fra 4 ohm e 32 ohm, a seconda del modello e delle dimensioni; la corrente che attraversa la spira è quindi calcolabile con la formula **I=V/R**.
Genuino non dispone di uscite analogiche e può solo pilotare con un segnale digitale che varia fra 0 e 5V, quindi su un altoparlante da 8 ohm (valore tipico degli altoparlanti) avremmo una corrente assorbita teorica pari a 5/8 = 625 milliampere. L'altoparlante farebbe lavorare la porta di Genuino al massimo della corrente erogabile (40mA) determinando una caduta di tensione significativa dei 5V nominali. Per non rischiare sovraccarichi dovremmo aumentare la resistenza del circuito aggiungendo in serie una resistenza da almeno 120 ohm, che porterebbe il totale a 128 ohm e a soli 39 milliampere l'assorbimento; in questo modo l'altoparlante non creerebbe problemi ad Genuino, ma non potrebbe produrre un volume elevato.
In soccorso di tutti i dispositivi che non si possono permettere correnti elevate sono stati sviluppati i trasduttori piezoelettrici, decisamente più limitati sul fronte della risposta in frequenza, ma capaci di generare alla frequenza di risonanza una buona pressione sonora pur con un assorbimento molto contenuto.
L'effetto piezoelettrico è manifestato da una serie di sostanze cristalline – come

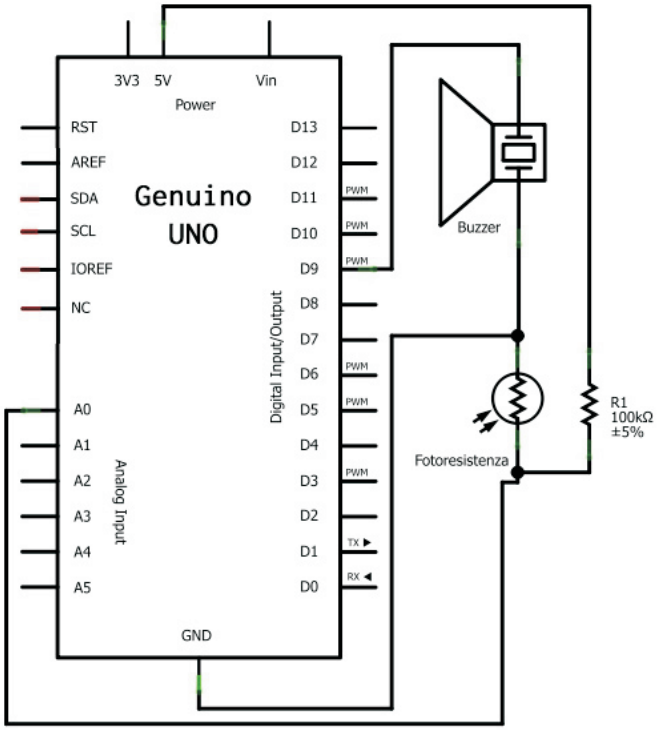
La misura dell'illuminazione

Il lux esprime il flusso luminoso per unità di superficie, con un lux che corrisponde a un lumen su un metro quadrato.
1 lx = 1 lm/m2
In pratica il lumen indica la quantità di fotoni emessi – o flusso luminoso – mentre il lux lo rapporta alla superficie che illumina. Poiché abbiamo come elemento l'area illuminata, nella vita reale entrano in gioco le caratteristiche delle lampade relativamente alla forma e all'angolo del cono di luce generato per passare dai lumen generati ai lux nella stanza. Per capire meglio, facciamo qualche esempio: se un quadro ha bisogno di essere illuminato con 50 lux per essere apprezzato al meglio ed ha una superficie di 2500 centimetri quadrati (50 x 50 cm), avrò bisogno di una sorgente da 12,5 lumen - ipotizzando che questa possa concentrare la propria luce esattamente sul quadro. Se il quadro fosse da 1 metro quadrato (100 x 100 cm) avrei bisogno di una sorgente da 50 lumen per mantenere la stessa intensità di luce sulla superficie del quadro e continuare ad apprezzarlo ben illuminato. Per ottenere da una medesima lampada più o meno lux su una superficie, basta quindi avvicinare o allontanare la lampada oppure stringere o allargare il fascio prodotto, ricordando che la variazione di lux segue il quadrato della variazione di distanza (dimezzo la distanza, aumento 4 volte i lux, mentre se raddoppio la distanza i lux diventano un quarto).
Qui di seguito alcuni esempi di intensità

Illuminazione	Esempio
10-4 lux	Luce delle stelle, cielo coperto
0,002 lux	Luce delle stelle e cielo limpido con riverbero
0,01 lux	Quarto di luna
0,27 lux	Luna piena in un cielo limpido
1 lux	Luna piena alle latitudini tropicali
3,4 lux	Limite scuro del crepuscolo civile con cielo limpido
50 lux	Soggiorno di casa
80 lux	Corridoio / bagni
100 lux	Giornata coperta molto scura
320–500 lux	Illuminazione di un ufficio
400 lux	Alba o tramonto in una giornata limpida
1.000 lux	Giornata coperta, tipica illuminazione di uno studio TV
10.000–25.000 lux	Piena luce diurna, ma non al sole

2011-2016 CC-BY-NC-ND



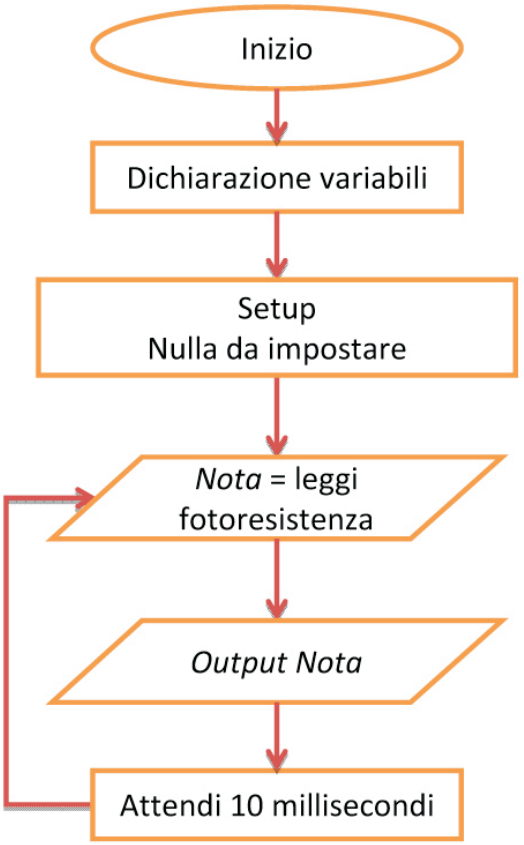


ad esempio alcune ceramiche prodotte dall'uomo, il quarzo e le perovskiti – e determina la variazione nelle dimensioni del cristallo, su un asse preciso, quando viene applicato un campo elettrico su due facce opposte del cristallo. Applicando un segnale alternato, si fa contrarre ed espandere il cristallo e questo movimento meccanico può essere trasformato in onde acustiche se trasferito a una superficie a contatto con l'aria (ad esempio la cassa dell'orologio da polso).

I trasduttori piezoelettrici disponibili in commercio sono di varie dimensioni e hanno varie frequenze di risonanza e tensione di funzionamento legati alla dimensione del trasduttore stesso. Proprio perché hanno una frequenza di risonanza, non sono l'ideale per riprodurre un'ampia gamma di frequenze, ma offrono una ottima resa se vengono fatti funzionare proprio alla loro frequenza di risonanza. La resistenza di un trasduttore piezoelettrico è di circa un migliaio di ohm alla frequenza di risonanza e quindi con i nostri 5V di alimentazione avremo un assorbimento di soli 5 milliampere. Purtroppo non otterremmo un elevato volume nemmeno in questo caso in quanto la tensione di lavoro ottimale è di almeno 10 volt. Negli orologi da polso il suono generato anche con i pochissimi volt disponibili è comunque sufficiente per essere sentito.

Una via di mezzo fra gli altoparlanti e i trasduttori piezoelettrici sono i buzzer magnetici, sempre caratterizzati da una frequenza di risonanza, ma costruiti con un sistema a spira magnetica che attrae un dischetto di metallo fissato a una

2011-2016 CC-BY-NC-ND



membrana non magnetica. Portando la membrana a cui è fissato il dischetto alla frequenza di risonanza si ha la massima pressione acustica pur contenendo l'assorbimento.

Il circuito elettrico

I componenti che ci servono per questo progetto sono solo tre: il trasduttore piezo, la fotoresistenza e una resistenza da 100 Kohm; il tutto va sistemato sulla breadboard e collegato con i ponticelli, come da disegno.

Il diagramma di flusso

Quello che vogliamo fare in questo progetto è molto semplice: dopo l'inizializzazione delle variabili, nel setup va impostato il pin di uscita per il piezo per poi arrivare al loop in cui leggiamo la fotoresistenza e utilizziamo il valore ottenuto come base per la generazione di una nota, quindi una breve pausa e si ricomincia con la lettura.

La generazione di suoni

La funzione **tone(pin, frequenza, durata)** che Genuino mette a disposizione permette di generare un'onda quadra con il 50% di duty cycle sul pin digitale specificato e per la durata specificata. La funzione prende in carico la generazione della nota senza bloccare l'esecuzione del programma e quindi se non si utilizza la funzione **noTone(pin)** o se non è stato messo il parametro di durata, per tutto il tempo in cui il programma sta girando – anche se sta eseguendo altre cose e non sta più chiamando la funzione tone() – dall'altoparlante continuerà ad uscire il suono con la frequenza definita.
Da un lato questo è un vantaggio perché ci permette di "fare altro" senza doverci occupare della generazione dell'audio, ma dall'altro ci obbliga a ricordarci di spegnere il suono se necessario. Questa funzione può essere attiva solo su un pin digitale alla volta, rendendo inefficace la funzione che imposta l'audio su un nuovo pin se non è stata prima disattivata la funzione sull'ultimo pin attivato con la funzione noTone(). In pratica non si possono generare più tonalità su pin diversi.
Un altro elemento da tener presente è **l'interferenza** che questa funzione ha sul PWM generato sui pin digitali 3 e 11.

Il codice

Le dichiarazioni iniziali – come nei casi precedenti – sono fatte solo per migliorare la leggibilità del listato, utilizzando delle variabili per il pin a cui è collegato il piezo (**D9**), quello in ingresso per la fotoresistenza (**A0**) e Nota per memorizzare la frequenza da generare con l'istruzione tone().

 2011-2016 CC-BY-NC-ND

```
1  int Piezo = 9;     // il trasduttore sul pin digitale 9
2  int Fotores = 0;   // la fotoresistenza sul pin analogico A0
3  int Nota;          // la variabile per la nota da riprodurre
4
5  void setup() {
6    // nulla da impostare
7  }
8
9  void loop() {      // iniziamo il nostro loop infinito
10   Nota = 3 * (analogRead (Fotores)) + 500; // leggiamo la fotoresistenza
11   tone(Piezo, Nota);     // suoniamo la nota che abbiamo calcolato sopra
12   delay(10);             // 10 millisecondi di pausa per consentire
13                          // al convertitore di leggere il valore
14                          // successivo nel modo corretto
15 }
```

Non abbiamo bisogno di definire nulla nel setup() in quanto i pin analogici sono già per definizione degli input e l'istruzione tone() non richiede che il pin utilizzato venga definito come output.

Ora proseguiamo con la parte **loop ()** dove leggiamo il valore della fotoresistenza con la funzione **analogRead (pin)** vista nel capitolo scorso e memorizziamo il valore in Nota, facendo però qualche aggiustamento cosmetico per migliorare le prestazioni audio del nostro progetto.

Il valore ottenuto dal convertitore A/D è compreso fra 0 e 1023, ipotizzando che la fotoresistenza sia in grado di variare da un valore molto elevato a uno molto basso. Nella realtà, dalle nostre prove, abbiamo misurato una variazione da 3 (piena luce) a 900 (buio). Utilizzare questi valori direttamente per generare un tono porterebbe a un netto sbilanciamento verso le basse frequenze. La frequenza più bassa generata possiamo quindi portarla a 500 hertz aggiungendo questo valore in un'espressione che somma 500 al valore letto. Se l'intervallo massimo è di circa 900 unità, saremmo poi limitati a questa piccola escursione di frequenza, ma anche qui basta un po' di matematica per migliorare le cose: se moltiplichiamo per 3 il valore letto, avremo 2700 Hz di gamma. Sommiamo i 500 visti prima e otterremo una gamma di frequenze da **500 a 3200 Hz.** Il tutto, in pratica, si ottiene con questa semplice espressione: "3*(valore letto)+500".

Ora che abbiamo il valore della nota, possiamo generarlo con l'istruzione tone () e procedere con l'istruzione successiva, ovvero 10 millisecondi di ritardo per consentire al convertitore analogico digi-

Commentare i listati

Avrete sicuramente notato come i nostri listati abbiano quasi ad ogni riga dei commenti , preceduti dal marcatore "//". Quando si scrive del codice, è molto importante utilizzare i commenti non solo perché si tratta di una "buona abitudine" da programmatore, ma anche perché con i commenti si rende tutto più facile e comprensibile. Con i commenti si possono anche temporaneamente annullare righe di codice, semplicemente mettendo all'inizio "//". Questo risulta particolarmente comodo per fare delle prove: aggiungendo e cancellando due soli caratteri per riga puoi evitare di cancellare e riscrivere per intero le varie righe.

Nel caso di commenti particolarmente corposi, oppure per sospendere intere parti di programma, si può usare il marcatore "/*" in apertura e "*/" ricordando però che questo metodo non può essere utilizzato se nel codice sono presenti dei commenti "//". Anche nei programmi più semplici, qualche commento aiuta a rendere evidenti i motivi per la scelta di un nome di una variabile o esplicita i collegamenti da fare a livello hardware e così via. Cercate di abituarti a metterne sin dai primi sketch e potrete anche voi apprezzarne l'utilità quando riaprirete dopo qualche tempo i vostri primi listati. I commenti nel codice sono anche un elemento fondamentale per la documentazione del codice stesso quando lo si vuole mettere in open-source: senza commenti il codice sarebbe difficilissimo da capire, anche se viene allegata della documentazione che ne spiega in generale il funzionamento.

2011-2016 CC-BY-NC-ND

tale di Genuino UNO di prepararsi a una nuova lettura. Se non lasciassimo questa pausa, otterremmo false letture dovute all'impossibilità del convertitore di riportarsi in uno stato adatto alla lettura di un nuovo valore. Potete verificare direttamente questo fatto non mettendo delay(10); nello sketch.
Come al solito, mentre lo scriviamo, facciamo attenzione all'inserimento dei punti e virgola al termine delle righe con funzioni e alle parentesi graffe in apertura e chiusura delle strutture.
Salvate lo sketch chiamandolo Capitolo_8 e proseguite con la verifica e il caricamento sulla scheda di Genuino UNO, già collegata con i vari componenti e con la breadboard.

Il collaudo

Dopo il solito lampeggìo dei LED TX ed RX a conferma del processo di caricamento, il trasduttore piezo si metterà ad emettere un suono un po' tremulo e dipendente dalla luce che colpisce la fotoresistenza. Avvicinatevi alla fotoresistenza con le mani e noterete come basta ben poco perché la nota emessa cambi. Provate a illuminarla con una torcia led e provate a oscurarla con un piccolo involucro.

Qualche esercizio

Anche con questo sketch è possibile fare un po' di pratica nella programmazione di Genuino affrontando alcuni semplici compiti da portare a termine. Ecco cosa vi suggeriamo per questo capitolo:

1. Invertite il rapporto fra tono e luce (adesso se la luce diminuisce, la frequenza aumenta: dovete ottenere il contrario).

2. Passate dalla nota continua a una nota staccata dalla successiva e di durata di almeno 100 millisecondi.

3. Create un effetto "spaziale" trasformando la singola nota in una breve scala da più note in sequenza, con la partenza che dipende dalla fotoresistenza.

4. Aggiungete alla versione 3 una soglia per silenziare l'audio se la luce è forte.

Un trasduttore piezo come sensore

I cristalli piezoelettrici si deformano se sottoposti a un potenziale elettrico e generano un potenziale se deformati. Questa simmetria nel comportamento permette di trasformare i trasduttori piezoelettrici in sensibili sensori acustici e di pressione. Nel mondo degli strumenti musicali, questi dischetti in ottone con uno strato di ceramica diventano quindi pick-up da applicare alla cassa armonica delle chitarre acustiche o trigger da inserire (opportunamente ammortizzati) all'interno dei fusti nelle batterie e dei tamburi.
Nelle nostre sperimentazioni, un trasduttore piezo va opportunamente collegato in parallelo a una resistenza di carico, indispensabile per non rischiare di danneggiare l'ingresso di Genuino e per avere delle letture in linea con quelle richieste per la propria applicazione. Se la resistenza è nell'ordine del migliaio di ohm, leggeremo solo i colpi secchi sul sensore, mentre se vogliamo un sensore per leggere la pressione, dobbiamo utilizzare una resistenza nell'ordine delle centinaia di migliaia di ohm. Va sperimentata la disposizione del sensore e il valore della resistenza in parallelo per trovare la soluzione migliore per il tipo di piezo scelto.

2011-2016 CC-BY-NC-ND

Spoiler...

Se non riuscite proprio a immaginare come affrontare gli esercizi indicati sopra, eccovi qualche suggerimento:

1. La frequenza varia secondo l'espressione "3*analogRead(Fotores)+500" che può assumere valori fra 500 e 3200 via via che la fotoresistenza aumenta la propria resistenza, e quindi il valore letto dal convertitore. Un modo "brutale" per cambiare il verso della variazione sarebbe quello di scambiare la resistenza con la fotoresistenza, mettendo quest'ultima fra l'ingresso e il positivo e non più il negativo. In questo caso andrebbero rivisti i parametri dell'espressione per gestire il diverso intervallo di valori prodotto in lettura. Un metodo più veloce è invece quello di rendere l'espressione corrente un valore da sottrarre alla frequenza più alta che vogliamo generare, attualmente 3200 Hz. In pratica, Nota diventa "3200-(3*(analogRead (Fotores))+500)".

2. Per questo esercizio, basta aggiungere una pausa con l'istruzione delay() e il comando di noTone(pin) seguito da una ulteriore pausa. A voi posizionare le funzioni al posto giusto, provando vari valori di delay().

3. Ecco una buona occasione per sperimentare con un ciclo for in cui ripetiamo la nostra nota addizionando un valore che aumenta a ogni passo del ciclo. Da tre a sei iterazioni possono essere sufficienti e il valore da aggiungere a ogni passaggio può essere fra 20 e 80.
   Il codice del ciclo for (inizializzazione; condizione; incremento) { } richiede un minimo di attenzione in quanto c'è una parentesi graffa "{" da aprire, quindi vanno inserite le istruzioni da ripetere, terminate su ogni riga dal ";" e al termine del ciclo non va dimenticata la "}" di chiusura.

Il vostro codice dovrebbe assomigliare a

```
9  void loop() {      // iniziamo il nostro loop infinito
10   Nota = 3 * (analogRead (Fotores)) + 500; // leggiamo la fotoresistenza
11   for (int i = 0; i < 7; i++)
12   {
13     tone (Piezo, (Nota + (50 * i)));
14     delay (10);
15   }
16   noTone(Piezo);
17   delay (50);
18 }
```

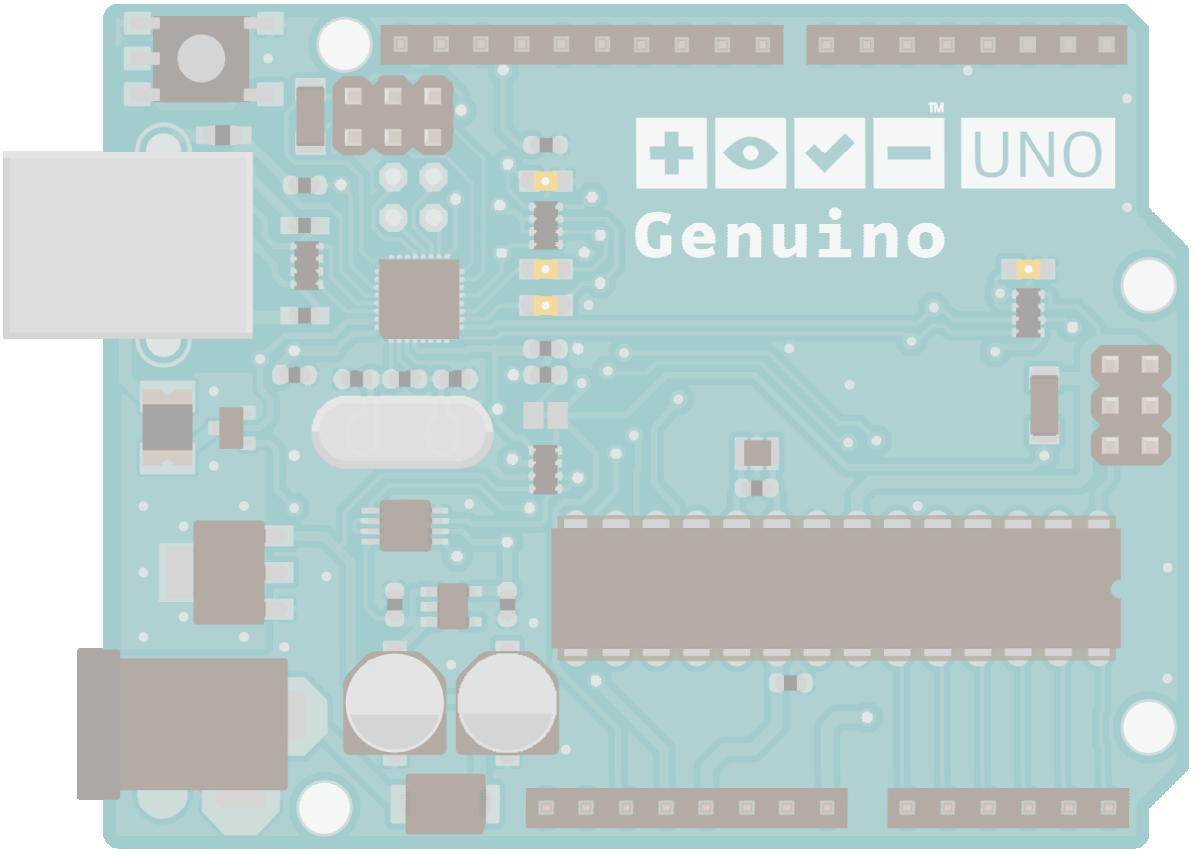

2011-2016 CC-BY-NC-ND

4. Per aggiungere questa funzione, puoi ricorrere a un controllo di tipo if .. else ricordando che se si verifica la condizione, si esegue il contenuto della struttura e si salta il contenuto di quella dopo else, mentre se non si verifica, viene eseguito il codice nella sezione di else. Partendo dal presupposto che non avete invertito la resistenza con la fotoresistenza e quindi con la luce forte diminuisce il valore letto dal convertitore, potete controllare se il valore di Nota scende sotto una certa soglia.
   Questo valore va scelto con qualche prova e nel nostro caso abbiamo trovato che fra 550 e 590 si ottiene una soglia ragionevole di illuminazione. Se Nota è inferiore alla soglia, allora "silenzio".
   Osservando il codice che abbiamo riportato per il terzo esercizio, notate come il ciclo sia seguito da noTone e da una pausa; questo significa che l'intera struttura di loop termina silenziando l'uscita. Se mettiamo il test sul valore di Nota prima di "suonarla" e mettiamo la parte che suona nella struttura else avremo ottenuto il nostro obiettivo. Allo stesso modo possiamo usare una struttura if semplice dove solo se Nota è maggiore della soglia, viene eseguita la parte che suona.
   A voi la scelta. Vi abbiamo suggerito la strada più lunga proprio perché si tratta di un esercizio e così avete imparato entrambe le versioni di if.

Ecco il codice con l'aggiunta della soglia:

```
9  void loop() {     // da qui tutto si ripete all'infinito
10   Nota = 3 * (analogRead (Fotores)) + 500; // Leggiamo il potenziometro
11   if (Nota < 590)
12   {
13     // non fare nulla
14   }
15   else
16   {
17     for (int i = 0; i < 7; i++)
18     {
19       tone (Piezo, (Nota + (50 * i)));
20       delay (10);
21     }
22     noTone (Piezo);
23     delay (50);
24   }
25 }
```


2011-2016 CC-BY-NC-ND



credit: Arduino.cc

2011-2016 CC-BY-NC-ND

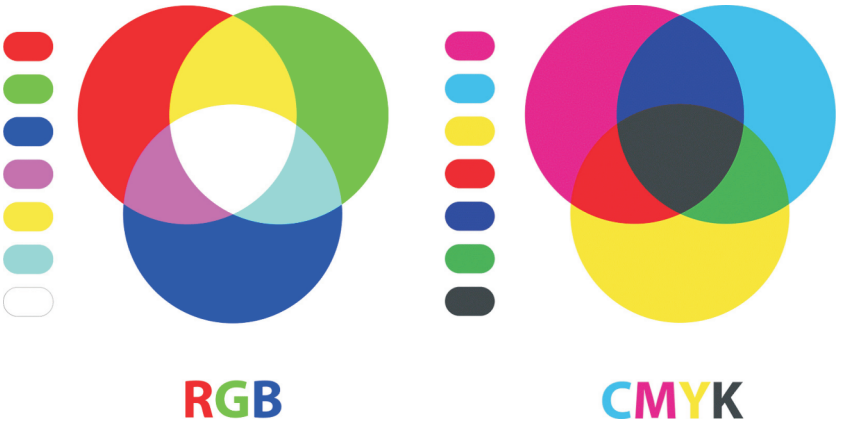
Il mondo è un arcobaleno di colori

Usiamo un LED RGB e il PWM per creare sedici milioni di colori in sequenza casuale

Da qualche tempo vanno di gran moda i piccoli gadget che sulle bancarelle serali nelle zone della movida cambiano colore in un arcobaleno di sfumature. All'interno di questi gadget si trovano dei singoli LED che ospitano nel loro involucro i tre chip Rosso, Verde e Blu con un integrato che li controlla. Più in grande e con strisce di numerosi LED gli architetti e i designer hanno vestito di luce con sfumature intense e affascinanti ogni genere di elemento architettonico: dalle fontane alle pareti, dalle suppellettili ai soffitti.
Tutti questi elementi hanno in comune due cose: l'utilizzo di LED più o meno potenti di colore rosso, verde e blu (in inglese Red Green Blue - RGB) e la tecnica di modulazione a durata dell'impulso (in inglese Pulse Width Modulation - PWM).

Anatomia del colore

Quando si parla di colori, grazie anche alla diffusione di fotocamere digitali e stampanti a getto d'inchiostro, siamo ormai abbastanza abituati a sentire termini come RGB e di CYMK. Le fotografie digitali sono con i singoli pixel in RGB, mentre la stampante utilizza quattro inchiostri per riprodurre su carta le varie sfumature.
Si tratta di due mondi complementari e legati fra loro, dei quali ormai sono stati sviscerati vantaggi e svantaggi, cercando e abbandonando negli anni potenziali alternative o reinterpretazioni aggiungendo ulteriori colori o variandone le tonalità.
Nella pratica, il modello RGB è per la **sintesi cromatica additiva**, mentre quello CYMK è per la **sintesi cromatica sottrattiva**; in altre parole, la sintesi additiva RGB somma fra loro luci di diversi colori, mentre quella sottrattiva usa dei pigmenti CYMK (Cyan, Yellow, Magenta, blacK - ciano, magenta, giallo e nero) per assorbire alcuni colori e rifletterne altri.
Il monitor a cristalli liquidi dei computer e dei televisori sfruttano dei tris di minuscole cellette - o pixel - rosso, verde e blu per generare dal nero (tutte spente) al bianco (tutte accese) passando da qualsiasi colore dell'arcobaleno purché entro certi limiti.
Se libro che state leggendo fosse su carta (e sappiamo che adesso lo state leggendo in PDF su un tablet o sul monitor di un computer), combinerebbe invece dei minuscoli puntini ciano, magenta e giallo - con l'aggiunta del nero dove opportuno - per partire dal bianco del foglio di carta e arrivare al nero sempre, passando dalle



I due modelli per la creazione dei colori: a sinistra la sintesi additiva RGB degli schermi che emettono luce e a destra quella sottrattiva della stampa tipografica.

2011-2016 CC-BY-NC-ND

sfumature dell'arcobaleno.
La complementarità dei due modelli appare più che mai evidente quando si effettuano da un lato e dall'altro le combinazioni dei colori due a due.

Sintesi additiva

*rosso + verde = **giallo***
*rosso + blu = **magenta***
*blu + verde = **ciano***
*rosso + verde + blu = **bianco***

Sintesi sottrattiva

*giallo + magenta = **rosso***
*giallo + ciano = **verde***
*magenta + ciano = **blu***
*giallo + ciano + magenta = **nero***

Nella stampa si aggiunge il quarto colore "nero" per diminuire il consumi di inchiostro, ottenere un migliore contrasto, un nero più neutro e aumentare il dettaglio. Senza nero si possono ancora stampare le sfumature e i grigi, ma questi ultimi sarebbero a rischio di qualche "dominante" se il bilanciamento fra i singoli CYM non è perfetto, anche considerando il bianco del foglio di carta su cui si stampa.
In entrambi i casi le sfumature si ottengono miscelando fra loro percentuali diverse di ciascun colore fondamentale, che nel caso della luce diventa l'intensità (da spento a 100%), mentre nel caso della stampa è la percentuale di copertura del pigmento su carta (da bianco, senza colore stampato, a 100%).

Qui sotto un esempio di stampa in quadricromia e nel dettaglio come i puntini della retinatura riescano a riprodurre le sfumature di colore.

Le sfumature in pratica

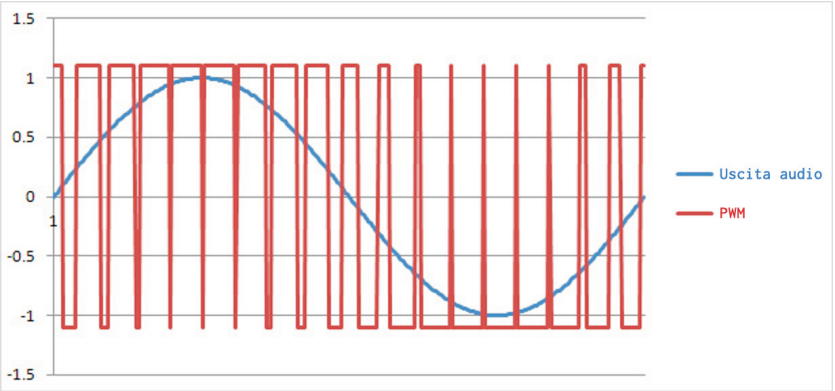
Se le sfumature dipendono dalla combinazione dei colori fondamentali in percentuali diverse, il numero di sfumature possibili dipende da quanto riusciamo a suddividere per ciascun colore l'intervallo 0 - 100%. Se ad esempio utilizziamo proprio la percentuale come valore (intero), abbiamo 100 diverse intensità per ciascuno dei colori e quindi il totale delle sfumature è 100*100*100, ovvero 1.000.000 di sfumature diverse. Fra queste avremmo solo 100 sfumature di grigio in quanto i tre valori variano all'unisono per ottenere dal nero al bianco

2011-2016 CC-BY-NC-ND

passando solo dai grigi.
Se aumenta il numero di passi per andare da 0 a 100%, aumentano conseguentemente le sfumature che possono essere generate. Con Genuino abbiamo una soluzione per variare l'intensità di ciascuno dei colori fondamentali con 256 valori, da 0 a 255. Questo porta il nostro conteggio a 256*256*256 che ha come risultato 16.777.216, ovvero quasi diciassette milioni di colori diversi.
Se siete appassionati di fotografia digitale, sapete anche che questo valore corrisponde al numero di sfumature che una fotocamera digitale è in grado di catturare e memorizzare in una immagine digitale, usando proprio tre byte per ciascun pixel.

E' una questione di media

Se avete seguito con attenzione i vari capitoli, avrete sicuramente notato che Genuino UNO non offre alcun segnale analogico in uscita. Questo in teoria ci impedisce di avere a disposizione un metodo per generare con i nostri LED dei livelli di intensità diversi da "completamente acceso" e "spento", eppure i circuiti "cambia colore" con Genuino sono presenti a decine in Rete e sui libri dedicati a questa piattaforma. Il segreto sta in un comportamento molto particolare del nostro occhio che, biologicamente, integra nel tempo gli impulsi luminosi.



Grazie all'integrazione del segnale, il PWM può essere utilizzato anche per gli amplificatori audio come avviene per quelli di "classe d" che offrono un ottimo rendimento e una bassa dissipazione di energia in calore.

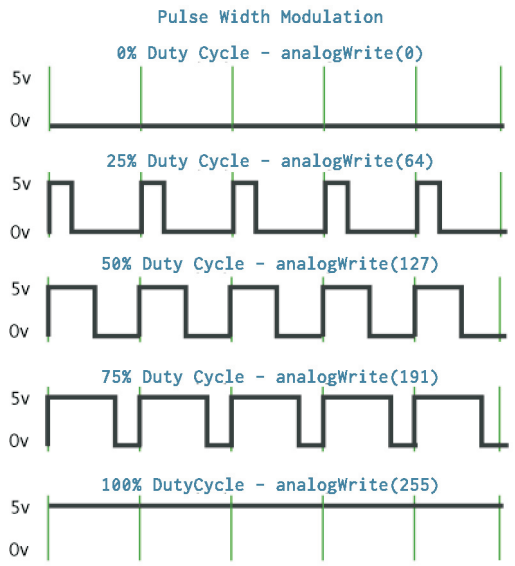
Possiamo immaginare questo "circuito biologico" come un secchio con un foro. Ogni impulso luminoso riempie un po' il secchio, che si svuota tramite il foro. La quantità di acqua nel secchio in qualsiasi istante equivale alla luminosità percepita e la quantità di acqua che fuoriesce dal secchio equivale alla soglia minima di luce percepita come tale.
Se riempiamo in un colpo solo il secchio con un po' acqua, percepiamo un lampo e se l'acqua è tanta, il secchio può richiedere un po' di tempo per svuotarsi. Questo fenomeno si chiama persistenza dell'occhio e dipende - nella fisiologia umana - da due fattori: intensità dell'impulso e frequenza di ripetizione dello stesso.
Proseguendo nel nostro esempio, se continuiamo ad aggiungere acqua, ma a piccole dosi, avremo che una volta raggiunta la quantità che il secchio è in grado di far defluire, l'acqua inizia ad accumularsi. La quantità di acqua nel secchio varia nel tempo tanto più visibilmente quanto le dosi sono grandi e distanziate, mentre tende ad essere meno "fluttuante" via via che le aggiunte diventano più frequenti e meno abbondanti. Portando il ragionamento al limite, se con una canna riempiamo il secchio, questo potrà riempirsi non appena la quantità che aggiungiamo

 2011-2016 CC-BY-NC-ND

Sopra: una fotografia scattata con un tempo di esposizione lungo e il risultato simula la persistenza nella visione dell'occhio.

Sotto: le forme d'onda generate dal PWM di Genuino a seconda del valore impostato con analogWrite().

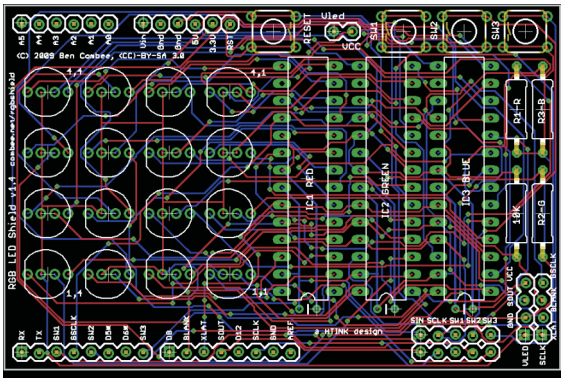
supera quella che defluisce dal foro di scarico.
Tornando ai nostri occhi, abbiamo l'esempio pratico della persistenza se qualcuno di fronte a noi disegna nell'aria con un punto luminoso, oppure osserviamo i disegni sul muro creati da un laser in una discoteca. Su questo fenomeno si sono basati i monitor a cinescopio e gli oscilloscopi , finché sono stati rimpiazzati dai cristalli liquidi e dalle altre tecnologie che non hanno più un "pennello" che dipinge un l'immagine come sequenza di linee ravvicinate.
Se il punto luminoso non si sposta, creando la scia, ma rimane stabile, abbiamo come effetto che la sua luminosità che varia nel tempo in base a quanto tempo sta acceso rispetto a quanto resta spento, il tutto rapportato alla sua luminosità assoluta. Spiegandolo in un altro modo, il nostro occhio fa la media nel tempo della luce e ci fa percepire la luminosità della sorgente come valore continuo proporzionato all'intensità istantanea e al rapporto acceso/spento.
Tutto questo, però, funziona solo a partire da una certa frequenza di lampeggio, altrimenti percepiamo il tremolio della sorgente assieme a un'intensità che ci sembra comunque inferiore a quella istantanea della lampadina accesa.
Con dispositivi come i LED, che non hanno praticamente inerzia nel cambio di stato fra acceso e spento, se non si sta nell'ordine di almeno un centinaio di hertz c'è la percezione del tremolio non appena si sposta lo sguardo dai LED. Potete verificare la cosa mettendo un LED con la resistenza di limitazione dell'assorbimento su un pin digitale e usando la funzione tone() per generare le varie frequenze.



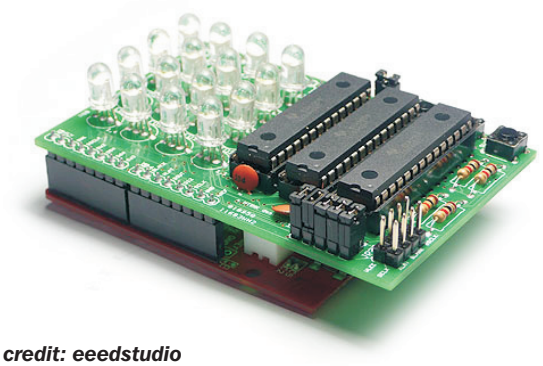
Il PWM di Genuino

Su una serie di pin digitali - nello specifico **sei** - Genuino UNO supporta il PWM, ovvero un'onda quadra - a circa 490 Hz - di cui è possibile gestire il rapporto fra semionda negativa e semionda positiva (il duty cycle) con un valore da 0 a 255. Grazie al PWM è quindi possibile accendere e spegnere un LED con una elevata velocità, gestendo il tempo in cui per ogni ciclo questo resta acceso.
Grazie all'effetto di integrazione dell'occhio descritto prima, questa variazione nel rapporto fra acceso e spento viene percepita come variazione di intensità della luce. Il risultato è

2011-2016 CC-BY-NC-ND



quindi un metodo efficace per gestire digitalmente 256 livelli di luminosità di LED su sei pin digitali.
Sui pin per i quali non è disponibile il PWM sarebbe possibile implementarlo via software, ma ottenere lo stesso livello di prestazioni su vari pin e con 490 Hz di frequenza è un'impresa quasi impossibile. Se servono più segnali PWM si possono utilizzare delle versioni diverse di scheda, come Genuino Mega2560 che ha 14 pin con PWM, oppure utilizzare dei circuiti integrati di supporto che gestiscono appositamente il PWM e sono gestibili da Genuino tramite una libreria dedicata (per esempio il TLC5940).



credit: eeedstudio

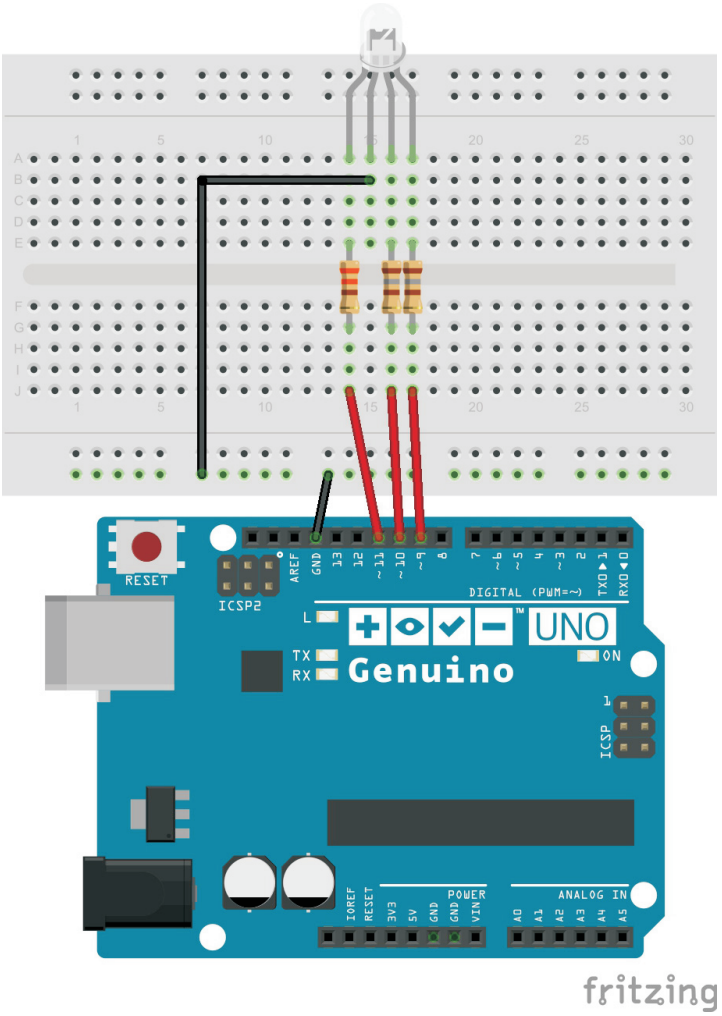
Sopra: lo shield RGB di Seeedstudio basato su tre chip TLC5940 per pilotare un totale di 16 LED RGB.

Tre LED in un solo involucro

Nel primo esercizio avete imparato ad accendere tre LED separati in sequenza, collegandoli ai pin 9, 10 e 11. Non si trattava di una scelta casuale, ma di un primo passo verso questo progetto che utilizza ancora gli stessi pin per pilotare tre LED, ma questa volta il package è di tipo speciale in quanto i tre LED sono racchiusi in un solo contenitore a quattro pin. Un pin è il catodo, comune ai tre LED, mentre gli altri tre pin permettono di pilotare l'anodo di rosso, verde e blu. La scelta della versione a catodo comune è stata fatta perché vogliamo accendere i LED utilizzando i 5V che ogni pin digitale produce nella parte attiva del duty cycle di PWM, mantenendo così un rapporto diretto fra valore del PWM e luminosità percepita (0=spento e 255= massima intensità).
Il piedino più lungo è il **catodo comune** e la sequenza dei collegamenti per il componente da noi utilizzato è **Verde** - **Catodo** - **Blu** - **Rosso**, ma può cambiare da LED a LED. Verificate quindi con il fornitore quale sia la corretta sequenza dei colori, oppure usate una resistenza da 330 ohm collegata ai 5V in uscita dalla scheda Genuino UNO, collegate il catodo a GND e controllate che colore si accende toccando ciascuno dei pin.
Come abbiamo visto nel quarto capitolo, a seconda del colore dei LED va utilizzata una resistenza di valore diverso; nel nostro caso il componente è un LED RGB che usa il verde puro, da 3,2 volt come il blu, mentre il rosso è da 2 volt. Le resistenze da usare sono quindi rispettivamente 180, 180 e 330 ohm per 10 milliampere di corrente a ciascun LED.
Se il LED che avete a disposizione ha l'involucro trasparente, potrebbe essere una buona idea renderlo satinato in qualche modo per favorire la miscelazione dei colori.

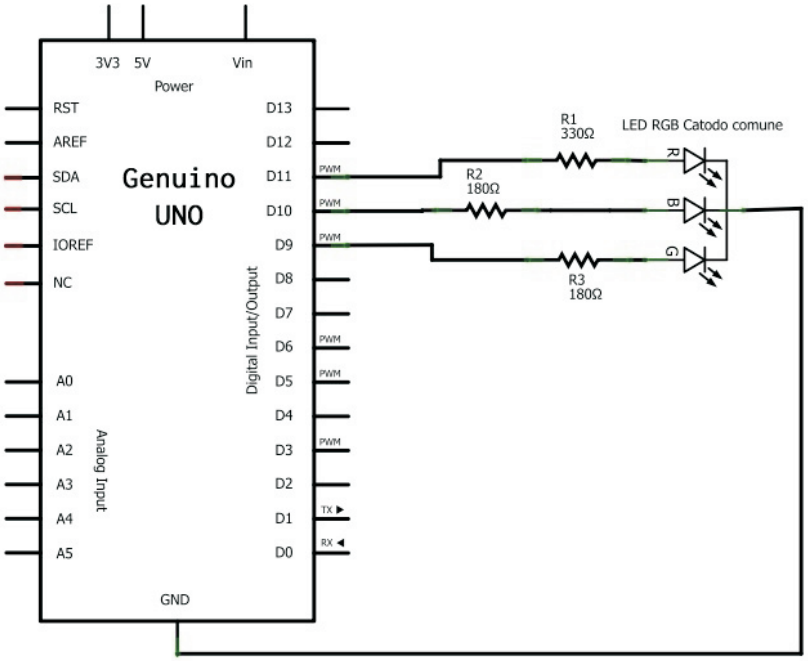
Sotto: dei LED RGB a catodo comune.

2011-2016 CC-BY-NC-ND



Il circuito in pratica

Utilizzando la breadboard possiamo collegare il nostro LED RGB alle tre resistenze. Solo una delle tre ha il valore di quelle utilizzate nel capitolo 6, mentre dovete utilizzare due resistenze da 180 ohm per verde e blu.
Tre cavallotti servono per collegare le tre resistenze che vanno agli anodi alle porte digitali D9, D10 e D11, mentre un cavallotto collega il catodo al pin GND. In questo schema di collegamento, il LED ha la sequenza Rosso - Catodo - Blu - Verde, quindi potrebbe essere necessario adattarlo al vostro LED RGB. In questo schema, il LED ha la sequenza Rosso - Catodo - Blu - Verde



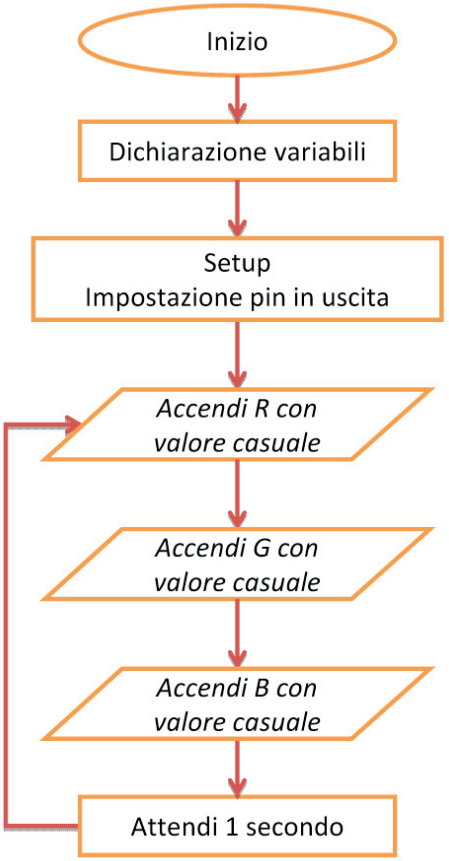
Il diagramma di flusso

Prima di avventurarci in formule matematiche per la gestione ragionata di colori e sfumature, vale la pena goderci un po' di colori casuali prendendo confidenza con il mondo di RGB e delle sue sfumature.
Il diagramma di flusso è quindi molto semplice in quanto descrive l'assegnazione dei pin ai vari colori, quindi nel setup c'è l'inizializzazione dei tre pin digitali in uscita e infine c'è il loop in cui scriviamo su ciascun pin, con la funzione analogWrite(), un valore casuale fra 0 e 255 e dopo un secondo di attesa ricominciamo.

La funzione per generare il PWM

Con analogWrite(pin,valore) attiviamo una particolare routine di Genuino che genera sui pin 3,5,6,9,10 e 11 un'onda quadra da 490 Hz con il duty cycle - variabile con il secondo parametro - fra 0 e 255. Il segnale non si ferma finché non interve-

2011-2016 CC-BY-NC-ND



niamo sul pin attivo in PWM con una istruzione di digitalRead() o digitalWrite(). Anche un analogWrite() con il valore a 0 porta il pin in questione al livello basso. Questa funzione **non richiede** necessariamente che i pin siano impostati come output, ma se intendiamo usare digitalWrite() per spegnere il pin dobbiamo prima aver definito i pin in OUTPUT.

I numeri casuali

Questa è la prima occasione che abbiamo per utilizzare la funzione che genera numeri casuali **random(min,max)**. Sulla casualità effettiva dei numeri generati potremmo aprire una lunga parentesi, ma vi basti sapere che si tratta di una sequenza che se avviata con un medesimo seme per il generatore di numeri casuali - funzione **randomSeed(valore)** - si ripete esattamente. Questo significa che se è necessaria una sequenza casuale che non si ripeta fra un avvio e l'altro del programma, dovremo usare randomSeed(valore) con valore che sia qualcosa di casuale - come una lettura analogica su un pin non collegato a nulla - prima di usare la funzione random(). Il valore min è opzionale e se non viene specificato è uguale a 0, mentre il valore max va sempre specificato, ricordando che non viene mai dato come risultato.

Il nostro sketch

Partendo dal presupposto che avete già verificato l'esatto abbinamento fra pin e colori nel LED RGB, possiamo procedere con la digitazione di questo sketch, breve, ma d'effetto. Abbiamo lasciato per "buona abitudine" le dichiarazioni della modalità dei pin nella parte di setup, anche se non obbligatorie per l'uso di analogWrite(). Come anticipato, queste sarebbero necessarie solo se volessimo usare anche la funzione digitalWrite() su questi pin.

```
1  int redpin =  11;   // pin a cui collegare l'anodo ROSSO
2  int greenpin = 9;   // pin a cui collegare l'anodo VERDE
3  int bluepin = 10;   // pin a cui collegare l'anodo BLU
4
5  void setup() {
6    pinMode (redpin,   OUTPUT);  // configuriamo i tre pin come uscite
7    pinMode (greenpin, OUTPUT);  // anche se non è richiesto per l'uso di
8    pinMode (bluepin,  OUTPUT);  // analogWrite()
9  }
10
11 void loop() {
12   analogWrite (redpin,   random(256));  // PWM casuale per ROSSO
13   analogWrite (greenpin, random(256));  // PWM casuale per VERDE
14   analogWrite (bluepin,  random(256));  // PWM casuale per BLU
15   delay(1000);
16 }
```


2011-2016 CC-BY-NC-ND

```
int redpin =  11;   // pin a cui collegare l'anodo ROSSO
int greenpin = 9;   // pin a cui collegare l'anodo VERDE
int bluepin = 10;   // pin a cui collegare l'anodo BLU

int R;              // variabile per il valore di ROSSO
int G;              // variabile per il valore di VERDE
int B;              // variabile per il valore di BLU

void setup() {
  pinMode (redpin,   OUTPUT);  // configuriamo i tre pin come uscite
  pinMode (greenpin, OUTPUT);  // anche se non è richiesto per l'uso di
  pinMode (bluepin,  OUTPUT);  // analogWrite()
}

void loop() {
  R = (((random(9))*32)-1);   // Valori casuali per RGB
  G = (((random(9))*32)-1);   // con step da 32 a cui è sottratto 1
  B = (((random(9))*32)-1);   // per non avere mai 256

  R = abs(R);
  G = abs(G);                 // elimino il segno per evitare "-1"
  B = abs(B);

  analogWrite (redpin,   R);  // PWM per ROSSO
  analogWrite (greenpin, G);  // PWM per VERDE
  analogWrite (bluepin,  B);  // PWM per BLU

  delay(1000);
}
```

Qualche variazione sul tema

Facendo funzionare questo sketch per qualche tempo vi sarete accorti che non tutti i colori generati sono per così dire "interessanti": sedici milioni di sfumature comprendono un sacco di colori scuri, poco incisivi, troppo vicini a un bianco sporco e così via. Forse è il caso di trovare qualche accorgimento per inserire degli elementi che privilegiano alcuni colori piuttosto che altri.

Il nostro suggerimento è quello di ridurre le sfumature per singolo colore fondamentale e inserire un passo negli incrementi possibili. Questo si ottiene sempre con la funzione random() a cui abbiniamo un'espressione che moltiplica il risultato per il nostro passo. Anche limitando a 8 variazioni per colore, avremmo ancora 512 sfumature, spaziate di 32 valori ciascuna. In termini di codice, il valore della luminosità potrebbe essere calcolato come segue:

valore= ((random(9)*32)

Il valore "9" non viene mai restituito, quindi avremo valori casuali fra 0 e 8 che moltiplicati per 32 diventeranno 0, 32, 64, 96, 128, 160, 192, 224 e 256. Questo è un piccolo problema perché 256 non è un valore valido per la funzione di generazione del PWM, ma perdere del tutto la massima intensità - portando a 8 il massimo della funzione random - non è una grande soluzione. Le possibili alternative sono varie e fra queste ti suggeriamo di fare un controllo con un if, dove se il valore è 256, lo decrementi di uno.

Un'altra possibilità è sottrarre sempre 1 al valore e usare la funzione **abs()** per trasformare -1, che sostituirebbe lo 0, in 1; abs() va applicato in una riga a sé stante perché potrebbe dare risultati inconsistenti se calcolato su espressioni. Se vi viene in mente altro, sperimentate pure. Da notare che sono state aggiunte le tre variabili R, G e B nelle dichiarazioni iniziali per non avere errori nel loop principale.

A voi il piacere di intervenire su questo sketch per creare ulteriori variazioni, magari escludendo i valori 32 e 64 che non risultano particolarmente visibili, ma possono "annacquare" gli altri valori (uguali quantità di R,G e B sono equivalenti a un bianco più o meno intenso).

2011-2016 CC-BY-NC-ND

E adesso un arcobaleno

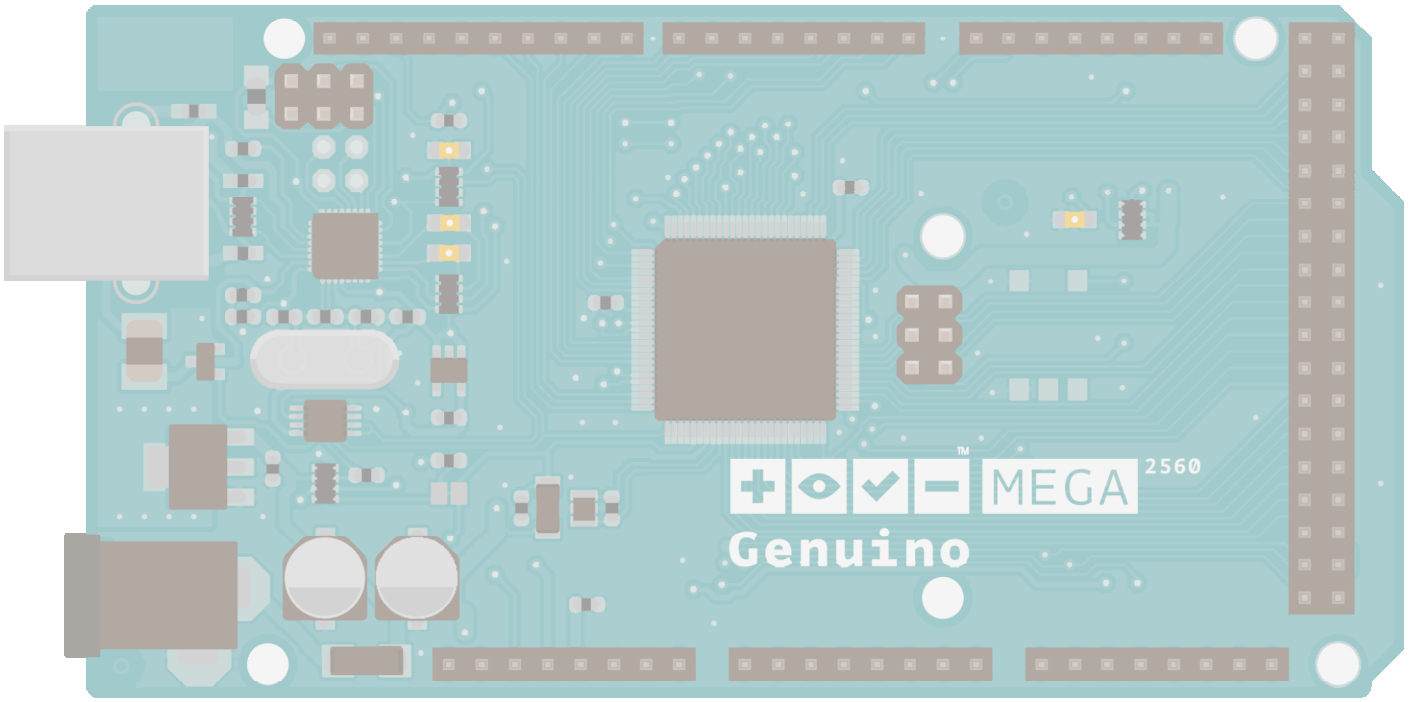
Visto che il circuito che avete cablato con la breadboard vi permette di controllare i valori di RGB, prima di smontarlo per passare al prossimo capitolo potete avventurarvi nella generazione di un arcobaleno, ovvero del passaggio da rosso a verde, da verde a blu e da blu a rosso passando per le sfumature intermedie. Il modo per farlo è – in linea di principio – abbastanza intuibile, ovvero si aumenta e si diminuisce l'intensità di ciascuno dei due colori per passare da uno all'altro. La situazione agli estremi di ogni coppia di colori è chiara: ne resta acceso uno solo. Meno chiara è la situazione a metà strada: dobbiamo sommare due valori massimi o due valori medi? Potremmo qui aprire un capitolo abbastanza ponderoso sulla percezione del colore e sulla sua rappresentazione con modelli diversi, ma per non annoiarti possiamo dirvi che la modalità più omogenea è quella che porta alla somma di tutte le componenti accese a 255. Se abbiamo al massimo 2 colori accesi in ogni istante, la somma delle loro intensità deve dare quindi 255.

Tradotto in termini di programma, se un colore primario si aggiunge ad uno già acceso, quest'ultimo va decrementato proporzionalmente. Niente di più facile: facciamo tre cicli da 0 a 255 e in ciascuno di essi facciamo salire un colore, mentre facciamo proporzionalmente scendere quello usato nel ciclo precedente.

```
1  int redpin =  11;   // pin a cui collegare l'anodo ROSSO
2  int greenpin = 9;   // pin a cui collegare l'anodo VERDE
3  int bluepin = 10;   // pin a cui collegare l'anodo BLU
4  int vel = 50;       // la velocità di transizione
5  int passo = 1;      // il passo con cui si incrementa ogni ciclo
6
7  void setup() {
8    for (int L = 0; L < 256; L += passo) { // da spento a rosso al massimo
9      analogWrite(redpin, L);
10     delay (vel);
11   }
12 }
13
14 void loop() {
15   for (int L = 0; L < 256; L += passo) { // da rosso a verde
16     analogWrite(redpin, 255 - L);
17     analogWrite(greenpin, L);
18     delay (vel);
19   }
20   for (int L = 0; L < 256; L += passo) { // da verde a blu
21     analogWrite(greenpin, 255 - L);
22     analogWrite(bluepin, L);
23     delay (vel);
24   }
25   for (int L = 0; L < 256; L += passo) { // da blu a rosso
26     analogWrite(bluepin, 255 - L);
27     analogWrite(redpin, L);
28     delay (vel);
29   }
30 }
```

All'avvio il programma non ha alcun colore attivato e quindi possiamo elegantemente preparare il primo colore con un ciclo crescente nel setup, poi i tre cicli "sali/scendi" si avvicendano all'infinito. Per rendere il tutto più piacevole e intrigante potete inserire due ulteriori elementi al programma: un ritardo per il passaggio da una sfumatura alla successiva e il salto di valori da fare fra una sfumatura all'altra. Provate anche a ragionare sul problema del massimo valore di PWM uguale a 255 e su come gestire il salto con i vari valori, quando questi non sono multipli di 255.

2011-2016 CC-BY-NC-ND



credit: Arduino.cc

2011-2016 CC-BY-NC-ND

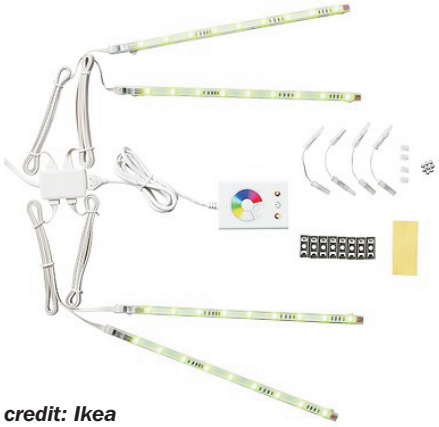
Verso una lampada per l'arcobaleno a comando

Usiamo un potenziometro per visualizzare tutti i colori dell'arcobaleno o per regolare l'intensità della luce bianca

In commercio ci sono vari dispositivi che permettono di gestire manualmente il colore di una lampada o di una striscia LED RGB agendo su un tasti di tipo digitale o su un potenziometro. Il precursore nel settore consumer per questo genere di prodotti è stata Philips con la bella - quanto costosa - lampada LivingColors - Da diverso tempo anche Ikea ha inserito nel proprio catalogo una soluzione meno sofisticata e più economica denominata Dioder, con un controller a filo dotato di una rotella di selezione del colore e due tasti per gestire degli automatismi.

Dietro a questi prodotti commerciali c'è l'applicazione del PWM su LED RGB, così come abbiamo visto nel capitolo precedente, tramite l'impiego di controller più o meno sofisticati. Il fatto che questi prodotti siano ora trattati da aziende come Ikea dimostra che si tratta di dispositivi richiesti e apprezzati anche dal grande pubblico: da un primo timido tentativo di diversi anni fa, ora Ikea ha diversi prodotti basati su LED RGB e le proposte di arredo che combinano questi sistemi di illuminazione ad altri prodotti quali scaffali e lampade sono sempre più presenti nelle esposizioni dei negozi Ikea; questo di fatto ha sdoganato i cambiacolore LED e li ha promossi a elementi d'arredo non solo per le camere dei ragazzi.

credit: Philips



credit: Ikea

Anche noi da tempo combiniamo strisce RGB con controller auto costruiti a elementi e strutture Ikea ottenendo risultati decisamente piacevoli e interessanti, sapendo finalmente che chi verrà a trovarci non ci guarderà più con occhio sospettoso, ma troverà questa illuminazione interessante e molto d'ambiente. Anni fa, invece, sarebbe stata vista come qualcosa di cattivo gusto o kitsch.

I due modelli per la creazione dei colori: a sinistra la sintesi additiva RGB degli schermi che emettono luce e a destra quella sottrattiva della stampa tipografica.

Da automatico a manuale

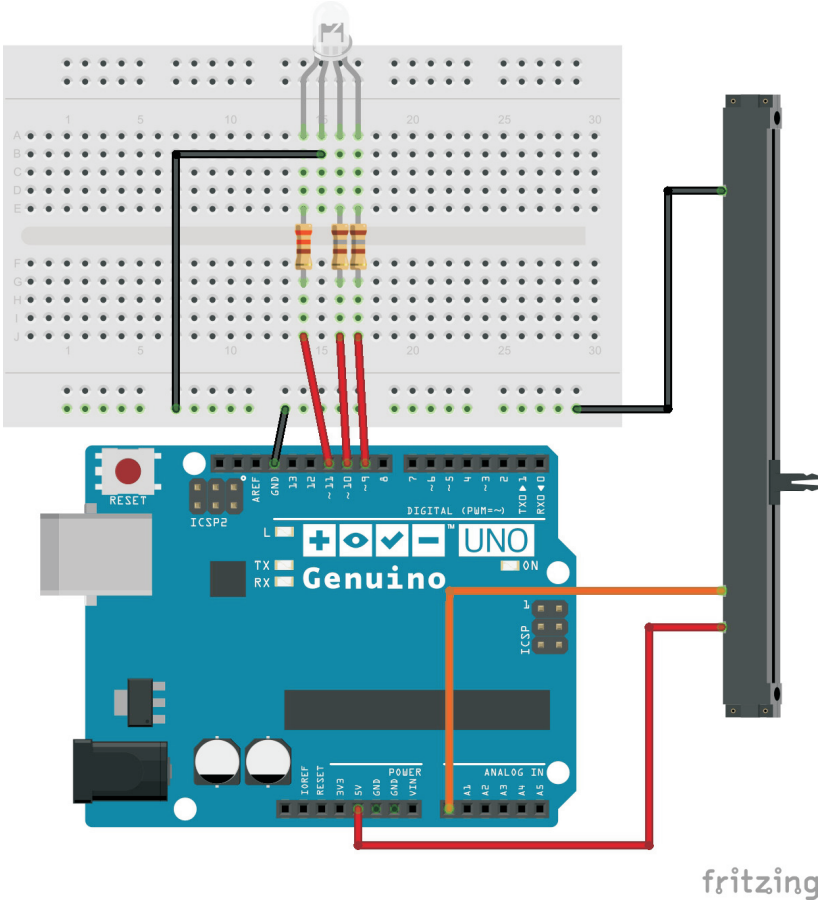
Nei capitoli precedenti ci siamo impadroniti di potenziometri, di valori analogici in ingresso, di LED RGB e di come si possono scrivere programmi per generare in modo automatico dei colori e degli arcobaleni.

Ora prendiamo tutto questo e lo ricombiniamo per gestire un generatore di colori manuale che determina il colore della combinazione RGB in base alla posizione dello slider di un potenziometro.

2011-2016 CC-BY-NC-ND

I prodotti commerciali con cui controllare le strip di LED RGB sono diventati molto comuni e anche relativamente economici. Il mercato cinese è la fonte principale di questi prodotti a basso costo.

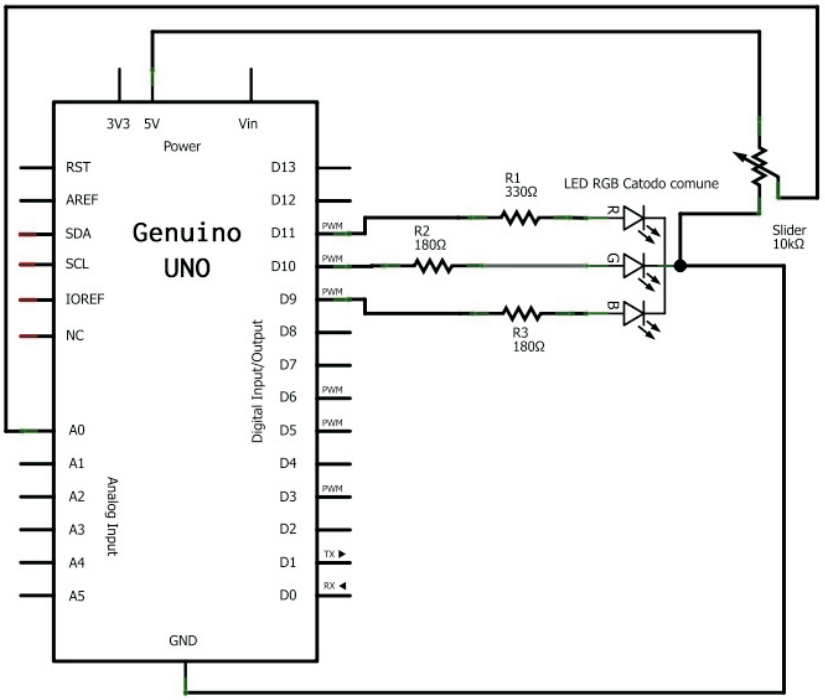
Dato che non ci piace fare le cose troppo semplici, aggiungiamo anche un altro elemento al progetto, ovvero il funzionamento del nostro "cambia colore" anche come lampada bianca di intensità variabile. In pratica, tre quarti del nostro slider influiranno sulla sfumatura di colore, mentre l'ultimo quarto attiverà la modalità "dimmer" dove tutti e tre i LED verranno attivati con il medesimo valore, variabile in base alla posizione dello slider, da tutto spento a tutto acceso. Mancherebbero solo due pulsanti "+" e "-" per agire sull'intensità del colori e avremmo di fatto una soluzione completa, ma lasciamo quest'ultima funzionalità alla parte degli esercizi in fondo al capitolo perché abbiamo già messo abbastanza carne al fuoco.



Perché anche il bianco?

Se pensate che aggiungere la parte di generazione del bianco a intensità variabile nel nostro progetto sia fonte di complicazione, siamo contenti di smentirvi e vi spieghiamo subito il perché. Dato che utilizziamo un potenziometro lineare sia come forma, sia come curva, siamo certi che i valori letti dalla funzione analogRead() variano da 0 a 1023; il programma che genera l'arcobaleno descritto nel relativo box del capitolo precedente utilizza tre cicli da 256 passi ciascuno per visualizzare l'intero arcobaleno. Se mettiamo assieme le due cose, notiamo che da una parte abbiamo 768 sfumature e dall'altra 1024 valori possibili. O scegliamo di rimappare matematicamente i 1024 valori su 768 o ne traiamo un vantaggio mettendo al lavoro i 256 valori che ci "avanzano". Come avrete capito, abbiamo scelto questa seconda strada che da un lato ci permette di evitare la matematica e dall'altro ci permette di avere una marcia in più.

2011-2016 CC-BY-NC-ND



I componenti del progetto

Prendiamo la breadboard predisposta per lo scorso capitolo e aggiungiamo il potenziometro a slider (potenziometro a scorrimento lineare) collegandolo a 5V e GND sui due terminali e ad A0 il cursore. Tutto il resto è già correttamente configurato e collegato.
Quando scegliete il potenziometro, ricordate che questo deve essere con curva lineare; lo sottolineiamo in quanto gli "slider", specie se trovati in un cassetto o recuperati su una bancarella sono più facilmente per l'audio e quindi logaritmici o antilogaritmici.

Impariamo a usare anche i bit

Visto che siamo in tema di didattica, vale anche la pena notare come i **10 bit** del convertitore A/D di Arduino UNO possano essere visti come **8 bit + 2 bit**, che rappresentano rispettivamente 256 e 4 valori. Il nostro codice potrebbe quindi usare i due bit più significativi per decidere in quale dei quattro quarti del potenziometro si trova il cursore e gli 8 bit meno significativi come valore da utilizzare per la generazione del colore o della sfumatura.
Trasformando i valori da **binario** a **decimale** ecco cosa otteniamo se dichiariamo i primi 2 dei 10 bit - detti anche i due bit più significativi, ovvero a sinistra - e lasciamo che i restanti otto abbiano qualsiasi valore:

binario	*decimale*
00nnnnnnnn	**0 - 255**
01nnnnnnnn	**256 - 511**
10nnnnnnnn	**512 - 767**
11nnnnnnnn	**768 - 1023**

Quattro zone, quattro blocchi funzionali

Ora che abbiamo trovato come avere una corrispondenza diretta fra le diverse posizioni del potenziometro e quattro intervalli numerici ben definiti, senza conversioni o approssimazioni, possiamo affrontare la stesura del nostro schema a blocchi.
Innanzi tutto decidiamo come gestire i due estremi del potenziometro: nell'ipotesi che questo progetto

 2011-2016 CC-BY-NC-ND

credit: www.lightbeaminc.com

Se i colori di questi LED vi sembrano diversi dal solito, avete ragione. Si tratta di una nuova tecnologia sviluppata da Light Beam Industries e denominata LED-C. L'involucro del LED contiene delle sostanze particolari che assorbono la luce emessa dal semiconduttore e la riemettono con colori specifici.

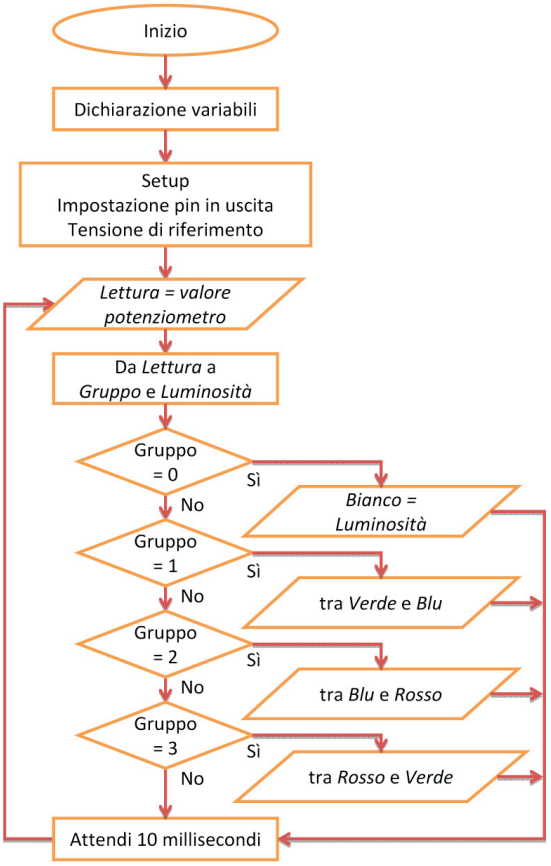
diventi un vero dispositivo per l'arredo domestico, dobbiamo prevedere qualche accorgimento che lo renda anche ergonomico. Se vogliamo fare a meno di un interruttore d'accensione, utilizzeremo un estremo a cui abbineremo il valore 0,0,0 di RGB, così da poter utilizzare quel fondo corsa come un virtuale spegnimento della lampada.

Facendo scorrere il cursore verso l'altro capo, avremo il bianco che aumenta di intensità fino al suo massimo, dopodiché si ha il passaggio alla modalità arcobaleno. Potremmo cominciare dal canonico rosso, passando dal verde, per arrivare al blu, ma il passaggio da bianco intenso a rosso e soprattutto viceversa potrebbe risultare poco piacevole, quindi pensiamo che la soluzione migliore sia tenere il bianco e il verde vicini: la sensibilità dell'occhio al colore ha il picco sul verde e quindi questo è il colore che l'occhio percepisce come il più luminoso dei tre fondamentali a disposizione, a parità di intensità luminosa.
Se iniziamo dal verde, proseguiremo con il blu, poi andremo sul rosso e concluderemo tornando nuovamente al verde. Il verde, quindi, apre e chiude il nostro arcobaleno, per questo lo ritroveremo anche all'altro estremo del potenziometro. Per chiarezza, precisiamo che i colori dell'arcobaleno possono essere ordinati in un anello chiuso e quindi interromperlo in un punto qualsiasi richiede che i due estremi siano praticamente uguali; in teoria potreste quindi scegliere un qualsiasi punto - o colore - dell'arcobaleno come inizio e fine, ma l'utilizzo di uno dei colori fondamentali rosso, verde o blu consente di scrivere lo sketch utilizzando direttamente gli otto bit di destra del valore a 10 bit in quanto allinea le routine di generazione dei colori con questi valori.
Riassumendo, ci sposteremo da 0 a 1023 con i seguenti intervalli di colore:

0 -> 255	**RGB 0, 0, 0 -> RGB 255, 255, 255**	**bianco**
256 -> 511	**RGB 0, 255, 0 -> RGB 0, 0, 255**	**da verde a blu**
512 -> 766	**RGB 0, 0, 255 -> RGB 255, 0, 0**	**da blu a rosso**
767 ->1023	**RGB 255, 0, 0 -> RGB 0, 255, 0**	**da rosso a verde**

Come abbiamo visto nell'esempio del capitolo scorso, la somma di ciascuna coppia di colori attiva nello specifico intervallo ha come somma 255 e quindi appurato che si va da un primo a un secondo colore, il valore della luminosità sarà da prendere per il primo colore dagli otto bit di destra del valore letto con analogRead(), mentre il secondo colore si otterrà sottraendo a 255 il valore del primo colore.

2011-2016 CC-BY-NC-ND



Il diagramma di flusso

Utilizziamo questo momento per mettere in ordine le nostre idee in una sequenza logica strutturata, approfittando anche del disegno dei blocchi per evidenziare la ripetitività della struttura di ciascun gruppo.
Nella fase di scrittura del codice ci preoccuperemo invece di trovare le soluzioni di programmazione più eleganti, compatte ed efficaci.

Le funzioni di mascheratura dei bit

Il primo ostacolo di programmazione da superare riguarda l'estrazione del numero del gruppo a cui appartiene il valore del potenziometro. Come abbiamo detto prima, i primi due bit più significativi sono quelli da isolare per individuare il gruppo. La funzione and "**&**" ci permette di mascherare i bit fra due valori secondo la funzione logica AND che ha la seguente tabella della verità:

0 AND 0 = 0
1 AND 0 = 0
0 AND 1 = 0
1 AND 1 = 1

Se vogliamo isolare i due bit più significativi sui 10 ottenuti dalla lettura del potenziometro (ricordate che il valore va da 0 a 1023, appunto ottenuto con 10 bit di campionamento), potremo effettuare un and con il valore decimale 768 - che in binario sui sedici bit dell'intero è 0000001100000000 - per ottenere solo questi quattro valori possibili: 0, 256, 512 e 768. Il passo successivo è più semplice e serve a portare il valore da 0 a 3 e si risolve dividendo per 256 il valore.
Questa divisione per 256 equivale anche a spostare a destra tutti i bit di otto posizioni, spostando i bit 9 e 10 nella prima e seconda posizione che, appunto possono rappresentare valori da 0 a 3; avremmo potuto ottenere il medesimo risultato con un'operazione singola - molto più da programmatori esperti - che è fare lo **shift** ("**>>**") a destra di 8 posizioni, scartando in un colpo solo i primi otto bit e restando con i due che ci interessano per il gruppo al posto giusto.
Proseguiamo isolando gli otto bit meno significativi per portare la variazione a valori fra 0 e 255, che si ripetono in sequenza quattro volte sull'intero percorso dello slider. Come abbiamo usato la funzione AND per isolare i bit 9 e 10, possiamo usare ancora un AND per isolare i primi otto bit. Il valore binario della maschera è 0000000011111111 - in decimale 255 -, quindi otterremo la luminosità dall'AND fra il valore letto dal potenziometro e 255.

 2011-2016 CC-BY-NC-ND

```
1  int redpin =  11;   // pin a cui collegare l'anodo ROSSO
2  int greenpin = 9;   // pin a cui collegare l'anodo VERDE
3  int bluepin = 10;   // pin a cui collegare l'anodo BLU
4  int Slider = 0;     // il pin analogico per lo slider
5  int Lettura = 0;    // dove memorizzeremo il valore del potenziometro
6  int Gruppo;         // da 0 a 3 per capire cosa fare con i LED
7  int Lumi;           // il valore di luminosità da usare con il PWM
8
9  void setup() {
10   pinMode (redpin,   OUTPUT); // Non serve con analogWrite(), ma
11   pinMode (greenpin, OUTPUT); // per farvi prendere l'abitudine
12   pinMode (bluepin,  OUTPUT); // ricordiamo che sono pin di USCITA
13   analogReference(DEFAULT);   // Il riferimento è impostato a 5V
14 }
15
16 void loop() {
17   Lettura = analogRead (Slider);  // in Lettura il valore del potenz.
18
19   Gruppo = (Lettura & 768) / 256; // scopriamo il gruppo nei bit 9 e 10
20   Lumi = (Lettura & 255);         // teniamo solo i primi 8 bit
21
22   switch (Gruppo) {
23     case 0:         // il primo intervallo da 0 a 255
24       analogWrite (greenpin, Lumi);   // tutti i colori assieme
25       analogWrite (bluepin, Lumi);    // con lo stesso valore per
26       analogWrite (redpin, Lumi);     // avere il bianco
27       break;
28     case 1:         // il secondo intervallo da 256 a 511
29       analogWrite (greenpin, 255 - Lumi); // Il verde dal massimo scende
30       analogWrite (bluepin, Lumi);    // mentre sale il blu
31       analogWrite (redpin, 0);        // il rosso resta spento
32       break;
33     case 2:         // il terzo intervallo da 512 a 767
34       analogWrite (greenpin, 0);      // il verde resta spento
35       analogWrite (redpin, Lumi);     // il rosso sale verso il massimo
36       analogWrite (bluepin, 255 - Lumi); // il blu scende dal massimo a 0
37       break;
38     case 3:         // il primo intervallo da 768 a 1023
39       analogWrite (greenpin, Lumi);   // il verde sale verso il massimo
40       analogWrite (redpin, 255 - Lumi); // il rosso scende
41       analogWrite (bluepin, 0);       // il blu resta spento
42       break;
43   }
44   delay (10);
45 }
```

Da una serie di if a switch case

Dal diagramma di flusso, si può supporre la necessità di ricorrere a una serie di if … else if … per gestire i 4 casi, ma abbiamo creato il valore del gruppo perché volevamo usare la struttura **switch (variabile) case valore** che risulta più ordinata e coincisa. La applichiamo come da manuale, inserendo il **break** alla fine di ogni blocco di case, così da passare subito al ritardo di 10 millisecondi quando viene eseguito un blocco case.

Un ritardo per stabilizzare

Dato che utilizziamo le funzioni di campionatura A/D e le uscite PWM, non possiamo reiterare la lettura del potenziometro troppo velocemente in quanto il convertitore ha bisogno di alcuni millisecondi per predisporsi a una nuova lettura. Senza il ritardo di una decina di millisecondi, lo sketch risulta instabile, con fluttuazioni nel-

2011-2016 CC-BY-NC-ND

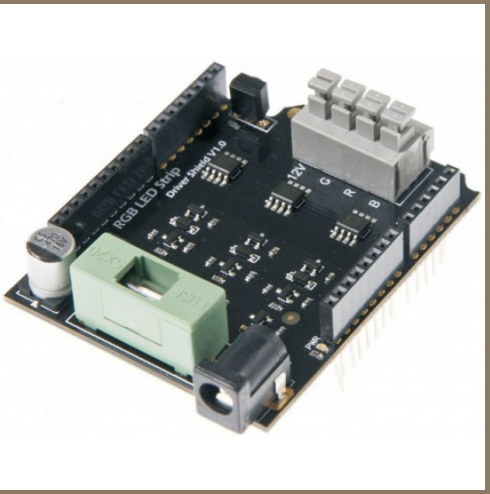
la lettura e qualche comportamento inconsistente sulle uscite PWM. A sketch ultimato, variate il valore della riga delay(10) per scoprire sotto a quale valore si manifesta l'instabilità.

Cambi di colore in velocità

Ogni blocco di variazione del colore non solo deve contenere le istruzioni di analogWrite() per i pin interessati dall'accensione, ma anche una che manda a zero il PWM del colore non interessato dal blocco. Questo è necessario per spegnere il terzo colore dato che non è possibile dare per scontato che il software riesca a leggere il passaggio da un blocco all'altro spegnendo un colore prima di passare a quello successivo. Se si sposta il cursore lentamente, il software legge tutti i valori e riesce a gestire le transizioni da un blocco colore al successivo o al precedente con i due valori a 0 e 255, ma se il passaggio avviene a salti e velocemente, è indispensabile azzerare il PWM del LED che non è interessato dai due valori gestiti nel blocco in cui si "atterra".
Ecco nella pagina il nostro sketch, che contiene anche alcune parti "opzionali" intenzionalmente inserite per chiarezza, come la definizione dei pin in uscita o l'impostazione del voltaggio di riferimento del convertitore A/D ai 5V di default.

Per metterci un po' di potenza

Se questo progetto vi sta ingolosendo e pensate che possa diventare qualcosa di più impegnativo, in termini di quantità di LED RGB che volete accendere, avete bisogno di aggiungere una parte di potenza.
Da veri auto costruttori potreste dotarvi di mosfet di potenza e qualche altro componente passivo per il corretto interfacciamento con Genuino, oppure potete restare lontani dal saldatore acquistando uno dei vari shield appositamente realizzati e che solitamente sono chiamati RGB Shield (Cercate su Amazon e ne troverete diversi).



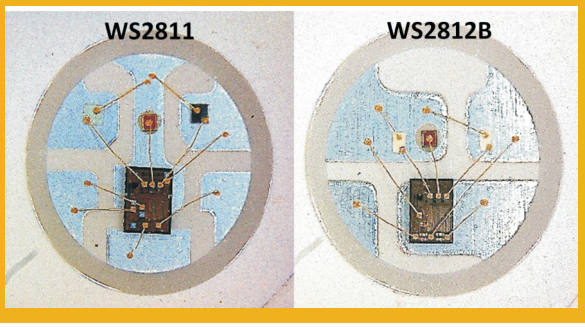
Come migliorare il progetto

All'inizio del capitolo avevamo suggerito di aggiungere un paio di pulsanti per variare l'intensità del colore, ma se avete seguito i vari ragionamenti, avete anche capito che non è semplice agire sui valori quando l'ambito di azione per l'aggiunta o la sottrazione è variabile e può anche essere di poche unità.
Una soluzione potrebbe essere quella di abbassare il livello complessivo del colore portando la somma dei due valori a 128 e utilizzando il nuovo spazio di valori fino a 255 per gestire la luminosità di ciascuna sfumatura. In questa operazione, ragiona sul fatto che un colore si ottiene anche dal rapporto che c'è fra i due colori fondamentali, quindi non è corretto incrementare il valore più basso quanto quello più alto, ma sarebbe da tenere in considerazione proprio il rapporto fra i due per incrementare ciascuno dei due valori preservando questo rapporto.
Se ad esempio abbiamo il rosso a 100 e il blu a 50, non possiamo aggiungere al rosso e al verde il medesimo valore in quanto il rapporto è di 2 a 1. Ogni volta che aggiungiamo 2 al rosso possiamo aggiungere 1 al blu e quindi possiamo al massi-

 2011-2016 CC-BY-NC-ND

Il futuro dei LED RGB

Nel campo dell'illuminotecnica i LED hanno fatto passi da gigante e ora troviamo nei negozi lampadine interamente a LED adatte all'utilizzo in casa al posto di quelle a incandescenza o al neon. Si sapeva che prima o poi i LED avrebbero sostituito le altre tecnologie, ma sapere che sono nei fari di automobile o nei lampioni in strada fa un certo effetto. Per quanto riguarda le strisce di LED RGB, con gli shield RGB possiamo controllare il colore, ma tutti i LED della strip possono assumere solo il medesimo colore in quanto collegati in parallelo (su ciascuno dei tre colori).
Sarebbe bello poter controllare il colore di ciascun elemento della strip, ma se solo un LED consuma tre pin in PWM, con una scheda Genuino UNO potremmo solo gestire due LED utilizzando tutti i sei pin dotati di PWM. Un controllo indipendente per ciascun LED RGB richiederebbe comunque un cablaggio molto complesso dove ogni LED richiederebbe dei collegamenti dedicati. Individuato il problema, gli ingegneri si sono subito messi all'opera sviluppando una serie di circuiti integrati capaci di gestire uno o piu LED RGB ricevendo dei comandi su una linea di comunicazione a due fili e capaci di ritrasmettere i comandi ad un altro circuito integrato identico, posto a valle su altri due fili creando un circuito a cascata. Questo sviluppo è andato avanti per circa due anni con varie soluzioni, poi è emerso fra i vari chip il WS2801: un driver a corrente costante su tre canali RGB, capace di ricevere i dati su un collegamento seriale a due fili (SCL e SDA) in ingresso e di trasmetterli su altri due collegamenti in uscita quando al suo ingresso se ne presentano di nuovi. In pratica si parla solo al primo WS2801 della sequenza e lui trasmette le informazioni al WS2801 successivo non appena riceve dei nuovi dati. I fili di colpo diventano 4 e le strisce di LED RGB diventano indirizzabili a livello del singolo LED RGB.
Già questo è stato un passaggio significativo, ma il colpo di scena è avvenuto quando qualcuno ha pensato di mettere sia i tre chip dei LED, sia il controller nel medesimo involucro 5050. Nasceva così il WS2811 e poi il WS2812B a quattro soli pin, con cui decine di migliaia di progettisti, artisti e designer stanno rivoluzionando i loro progetti proprio grazie a questa inaspettata flessibilità
Piccoli, ma facili da gestire con tre soli fili – di cui due per l'alimentazione – questi LED singolarmente indirizzabili stanno scatenando la fantasia di molti utenti e altrettanti produttori. Basta fare una ricerca in Google con questa sigla per scoprire centinaia di soluzioni e migliaia di progetti. Ovviamente con questi nuovi LED, Genuino UNO riesce a fare cose meravigliose. Vi invitiamo quindi a procurarvene qualcuno e iniziare con essi la vostra prima avventura con le librerie: un aspetto dell'Arduino Software (IDE) molto speciale che vi permette di pilotare o gestire i dispositivi più disparati mantenendo la semplicità che avete sin qui sperimentato. Le librerie che pilotano questi LED sono Adafruit NeoPixel e Fast_LED.



mo arrivare a 255 di rosso e 127 di blu.
Lo stesso ragionamento vale per il decremento verso il minimo.
Se vi interessa sperimentare in questa direzione, potete iniziare creando una fluttuazione automatica limitata a una ventina di valori, mettendo a punto l'espressione che calcola il rapporto fra i due colori fondamentali e lo applica agli incrementi dei due valori, sia in aumento che in decremento.
Più semplice è l'aggiunta di un automatismo per avere il passaggio automatico su tutti i colori dell'arcobaleno quando si preme un tasto. Si tratta infatti di dirottare l'assegnazione del valore di Lettura dal convertitore A/D a un ciclo che carica i valori da 256 a 1023 se è stato premuto il tasto e impostata a 1 la variabile definita per lo scopo. Ovviamente devi gestire il pulsante nel modo opportuno, attivando o disattivando il ciclo automatico ad ogni pressione, senza leggere come pressioni multiple un tasto semplicemente premuto a lungo.

2011-2016 CC-BY-NC-ND

Servomotori per la robotica

Sempre con il PWM e un potenziometro possiamo realizzare un embrione di arto meccanico grazie a un servomotore

Ogni anno, in varie parti d'Italia, si svolgono diverse manifestazioni che hanno la robotica e l'automazione hobbystica al centro dell'attenzione. Chi partecipa con i propri robot e sistemi a controllo remoto fa spesso parte del mondo della ricerca e delle università, ma non mancano anche le scuole di grado inferiore e gli appassionati. Con sempre maggiore frequenza il cuore di queste realizzazioni vede la piattaforma Arduino e le sue declinazioni specializzate, a dimostrazione di quanto il progetto Arduino abbia una forte vocazione robotica.

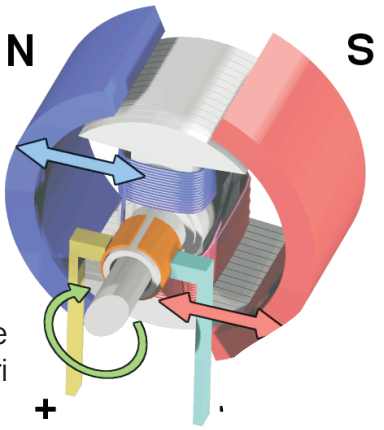
Sensori e attuatori

Nella robotica, l'intelligenza artificiale e gli algoritmi sono solo una parte di ogni progetto, mentre i sensori e gli attuatori forniscono le capacità di interagire con l'ambiente circostante con vari livelli di abilità. Nei capitoli precedenti avete imparato a leggere i valori di luce con una fotoresistenza, a interpretare una rotazione su un asse con un potenziometro o a rilevare una posizione assoluta su un percorso lineare grazie ad uno slider. Un discreto arsenale che potrete mettere all'opera assieme ad altri sensori più sofisticati e destinati alla rilevazione della distanza, dell'accelerazione e di altri parametri ambientali statici e dinamici. Il tutto ovviamente già pronto e supportato da librerie dedicate.
Gli attuatori sono invece dispositivi elettromeccanici che permettono di muovere con varie modalità le parti del nostro robot o addirittura di spostare l'intero robot.
Fra gli attuatori, i motori sono quelli più diffusi e utilizzati. Con i motori si possono fare tante cose, anche in base al loro tipo: in continua, stepper o servo.

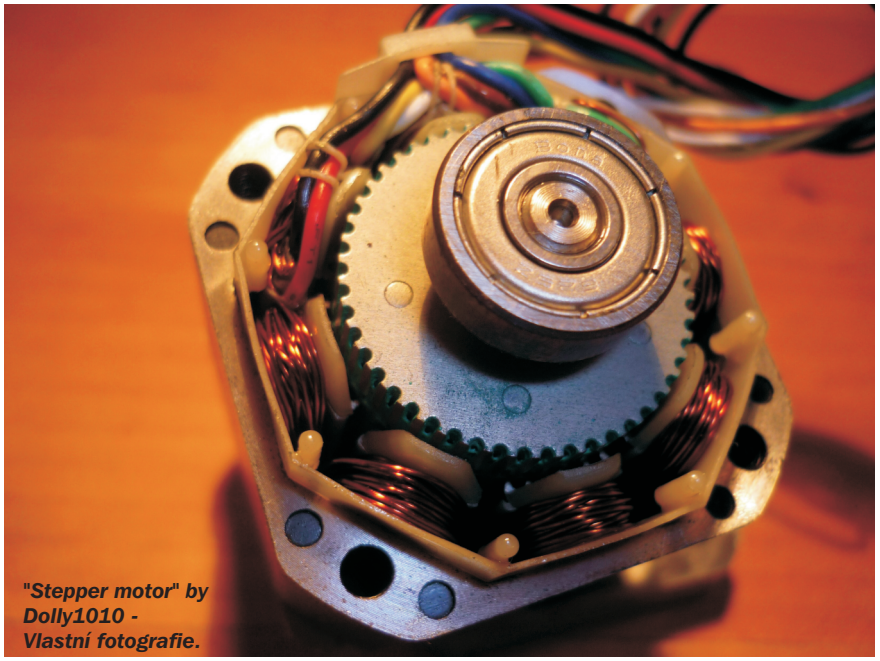
I motori in continua

Tutte le macchinine e i modellini elettrici più semplici utilizzano questo tipo di motore, caratterizzato dalla velocità di rotazione che dipende dalla tensione di alimentazione: più tensione = più velocità, con ampi margini di manovra.
Ne esistono di svariate misure, con e senza spazzole, ma sempre caratterizzati da magneti permanenti nella parte esterna ed elettromagneti solidali al rotore.
I più piccoli sono quelli che fanno ruotare delle minuscole masse eccentriche nei sistemi di vibrazione dei cellulari, mentre negli scooter elettrici ne trovate di molto più grandi e potenti, in grado di generare abbastanza spinta da spostare una persona su strada a diversi chilometri all'ora.
Il comportamento di questi motori è quindi molto semplice: applico una certa tensione e il motore gira a una certa velocità. Se invece vogliamo sapere quante rotazioni sono state fatte o la velocità di rotazione dell'asse per effettuare dei calcoli, dobbiamo aggiungere un sensore che conta i giri dell'asse o qualcosa di simile.
Con questi motori si possono anche realizzare dei sistemi che usano viti

Sotto: lo schema di un motore in corrente continua a magneti permanenti.



2011-2016 CC-BY-NC-ND


"Stepper motor" by Dolly1010 - Vlastní fotografie.

Uno stepper motor aperto in cui sono ben visibili gli avvolgimenti esterni e il rotore con i magneti permanenti; da notare i dentini del rotore, necessari per il funzionamento a passi.

senza fine e ingranaggi per aprire e chiudere pinze o flettere articolazioni, rilevando l'aumento dell'assorbimento del motore per capire quando si arriva a inizio o fine corsa.

Motori stepper

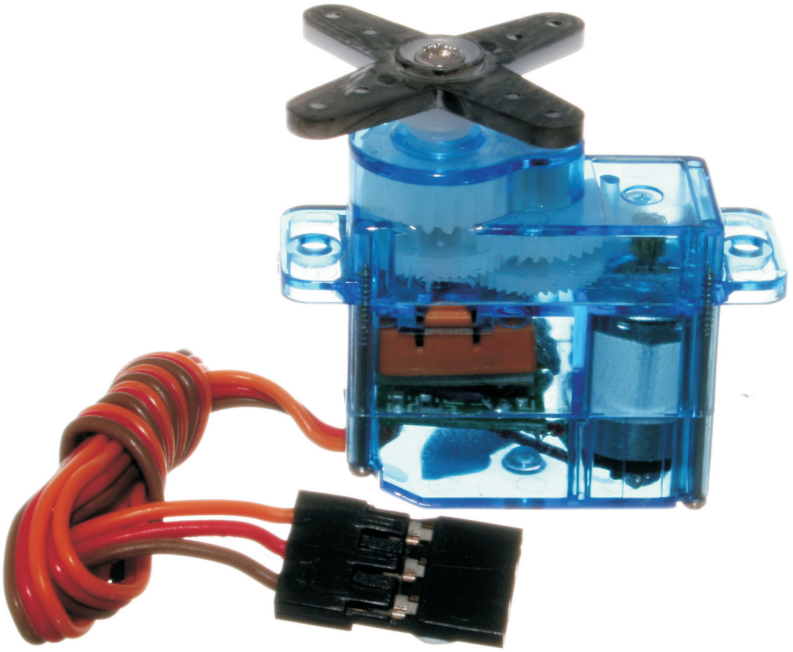
Nati per una gestione controllata del movimento di ingranaggi e pulegge, gli stepper sono dei motori di potenza caratterizzati da una serie di avvolgimenti che gli permette di ruotare il proprio perno con un meccanismo a singoli passi. Le caratteristiche di questi motori sono quindi completamente diverse e riguardano il numero di "step" per ciascuna rivoluzione completa assieme alla forza che il motore esercita con ogni passo. Oltre a questo, gli stepper sono motori che vengono apprezzati anche per la loro capacità di "bloccare" e tenere una particolare posizione, con accelerazioni e frenate decisamente brusche. Dispositivi come i CNC, i plotter e gli altri sistemi robotizzati a controllo numerico trovano in questo tipo di motori la soluzione ideale: ogni passo ha una precisa sequenza elettrica da eseguire e risulta quindi perfettamente controllabile; con sistemi abbastanza semplici è quindi possibile gestire gli stepper controllandone il movimento a livello di singolo step in ciascuna delle due direzioni. In sistemi meccanici di precisione gli stepper consentono sia di gestire il numero esatto di passi di rotazione, sia la velocità di rotazione dell'asse. Come contropartita, gli stepper sono abbastanza impegnativi sul fronte elettrico, con assorbimenti di corrente sostenuti, anche per le versioni più piccole.
Data la loro notevole diffusione all'interno di varie macchine elettroniche di tipo consumer (per esempio le stampanti laser e quelle a getto d'inchiostro), è facile trovarne di seconda mano e surplus alle fiere di elettronica, sui banchetti specializzati in materiale di recupero. La documentazione per il loro interfacciamento è facilmente reperibile in rete, mentre non potevano mancare vari shield già pronti per il collegamento e il pilotaggio di questi motori.

I motori servo

Chiudiamo la carrellata dei motori con quelli che utilizzeremo anche per il nostro progetto, ovvero i servomotori, o "servo".
La loro origine è squisitamente modellistica e quindi hanno caratteristiche quali l'alimentazione a basso voltaggio e una forma di gestione di tipo assoluto per la

2011-2016 CC-BY-NC-ND

posizione. Su aeromodelli, macchinine e altri modellini radiocomandati, i servomotori hanno il compito di gestire il posizionamento e il controllo delle parti che solitamente influiscono sulla velocità e sulla direzione. Sugli aeromodelli ad esempio, il timone, i flap e le altre superfici di controllo vanno regolate con precisione; sulle macchinine, lo sterzo e - se a motore - il gas, sono gestiti ancora con dei servo.
Il controllo avviene tramite un'onda quadra in cui la durata dell'impulso positivo - variabile solitamente fra 544 e 2400 microsecondi - determina la posizione assoluta del braccio rotante.



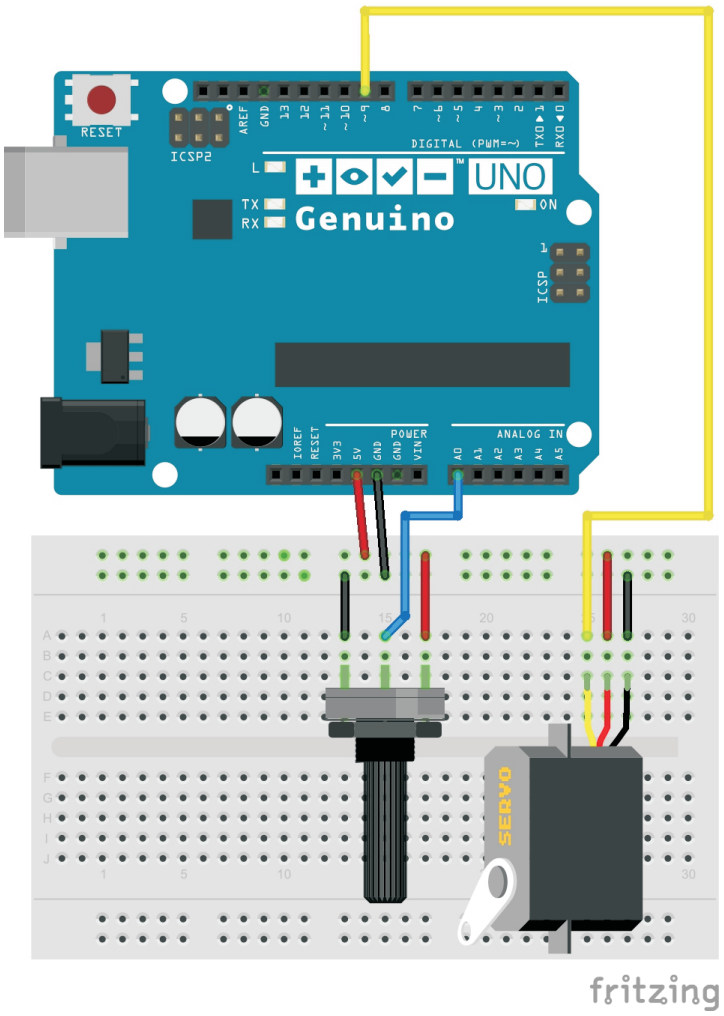
Il servomotore qui sopra ha il case trasparente e permette di vedere sia la parte elettronica, sia la serie di ingranaggi che demoltiplica la rotazione del piccolo motore in continua.

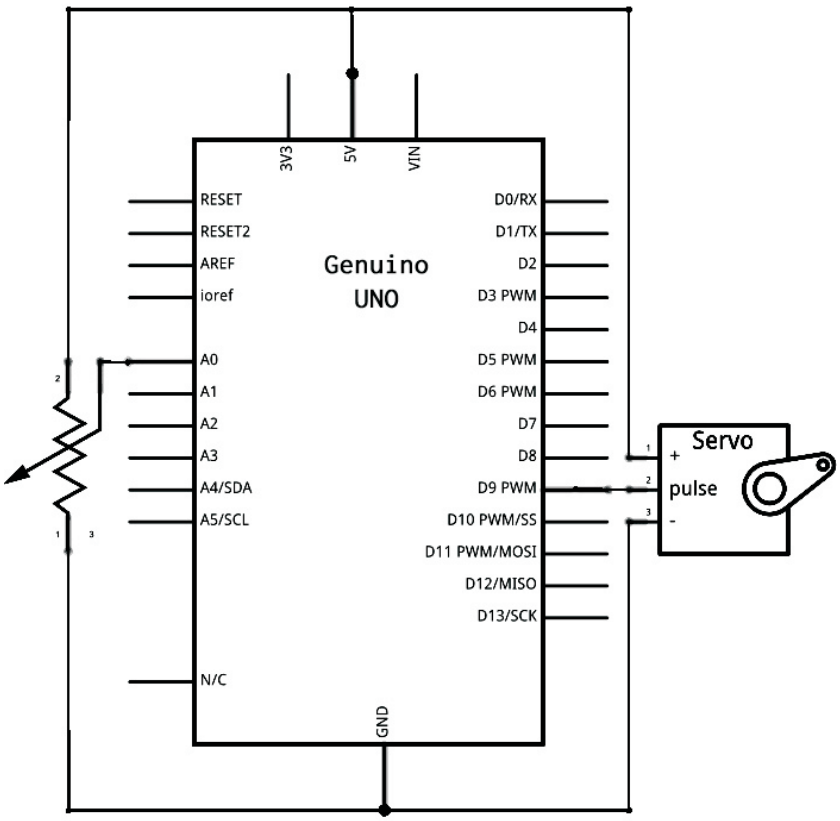
Normalmente i servomotori possono ruotare di 180 o più gradi, oscillando da una parte all'altra, senza mai effettuare più giri su se stessi. Nel caso di servo che invece possono compiere più rotazioni, il valore dell'impulso determina la velocità di rotazione.
Praticamente l'ideale per essere gestiti da Genuino e infatti per i motori servo è stata addirittura scritta una libreria ufficiale in grado di pilotarne - come vedremo più avanti - ben 12 sulla maggior parte delle versioni di scheda.
Anche se a prima vista 12 motori possono sembrare molti, è importante sottolineare che in robotica i servo non bastano mai: ogni "grado di libertà" (che equivale grossolanamente a un movimento su un unico asse) richiede un motore e quindi ogni rotazione e ogni snodo impegnano una risorsa.
Tornando ai servomotori, la loro anatomia è abbastanza complessa in quanto contengono sia dei circuiti elettronici di controllo, sia una parte meccanica abbastanza articolata. L'elettronica ha il compito di posizionare con precisione gli ingranaggi in base alla durata dell'impulso, mentre la serie di ingranaggi interna ha il compito di demoltiplicare la rotazione del perno di un normale motorino in continua ed ottenere così una forza di torsione elevata. Gli ingranaggi svolgono anche una funzione di "freno di stazionamento" anche senza alimentazione.

Il progetto

Dopo tutti i vari esperimenti che vi abbiamo fatto fare, abbiamo pensato di concludere con una semplice applicazione che vi permette di posizionare il braccio di un servomotore in accordo con il valore di un potenziometro.
Dal punto di vista educativo, vi spiegheremo come utilizzare una libreria esterna

2011-2016 CC-BY-NC-ND





per aggiungere delle funzionalità specifiche al linguaggio dell'Arduino Software (IDE), uno degli ultimi passi da fare per avere una ragionevole infarinatura nella programmazione di Genuino.
Come collocamento nel mondo reale del progetto, possiamo dirtvi che l'abbinamento fra un sensore e un attuatore è alla base di vari sistemi di telepresenza, dove un uomo manovra un braccio dotato di sensori e a distanza c'è un braccio che tramite motori replica esattamente il movimento. Non avendo a disposizione le parti meccaniche necessarie a trasformare il potenziometro e il servo in due bracci articolati, dovrete accontentarvi della dimostrazione in linea di principio.

La libreria Servo

Quando avete installato l'Arduino Software (IDE) usando l'installer o estraendo il file compresso in una serie di cartelle, avete anche estratto le librerie ufficiali sviluppate dal Team per gestire, solo quando necessario, dei dispositivi specifici. Dato che la memoria per il codice da eseguire non è illimitata, l'impiego di librerie ha permesso di limitare lo "spreco" da codice inutilizzato al minimo. Quando serve, si include la libreria e immediatamente nell'IDE diventano disponibili le funzioni aggiuntive. In fase di compilazione e produzione del codice oggetto, verranno utilizzate solo le funzioni di base e quelle delle librerie incluse esplicitamente, ottenendo sempre la miglior combinazione fra funzionalità del codice e occupazione di memoria.
La libreria Servo implementa sei funzio-

2011-2016 CC-BY-NC-ND

ni specifiche e aggiunge anche una classe "servo" con cui va assegnato un nome identificativo a ciascun motore che intendete utilizzare. Praticamente definiamo un oggetto come tipo "servo" e questo serve dal punto di vista sintattico per le funzioni specifiche.
Sono quindi espressioni valide

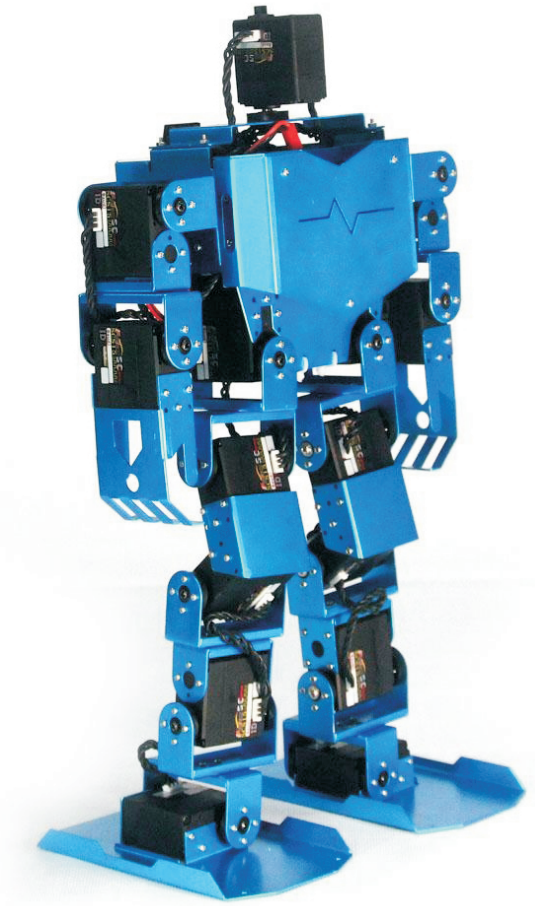
Servo Motore1;
Servo mioServo;
Servo gomito;

con le quali abbiamo creato gli oggetti Motore1, mioServo e gomito che ci permettono di costruire le successive funzioni

attach()
write()
writeMicroseconds()
read()
attached()
detach()

La prima - servo.attach(pin,min,max) - va usata per collegare a uno specifico pin digitale il servomotore corrispondente al nome servo assegnato precedentemente.
Per esempio

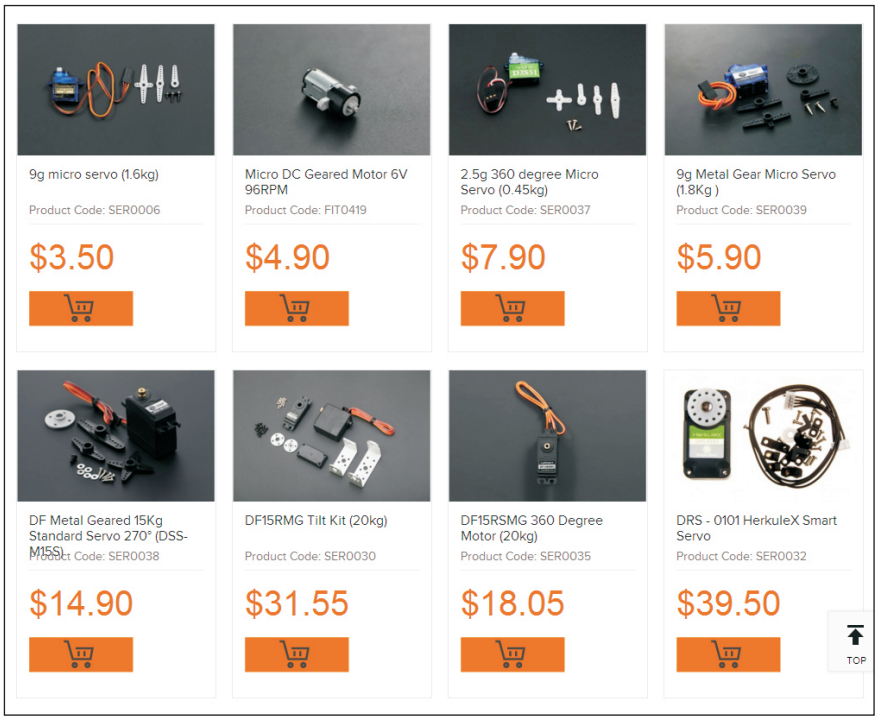
Servo gomito;
…
gomito.attach(9);
…



Qui sopra un robot realizzato con 17 motori servo, un trasmettitore basato su una scheda Genuino Micro compatibile e dei controller di servomotori con BUS UART. Prodotto da Feetech RC Model Co.

Trattandosi di una impostazione che potrebbe restare valida per l'intero sketch, può essere opportuno metterla nella parte di setup() del codice. I due valori min e max sono opzionali e indicano i valori minimi e massimi di durata dell'impulso in microsecondi richiesti dal servomotore per raggiungere effettivamente il minimo e il massimo della propria escursione di movimento.
Con **servo.write(gradi)** passiamo alla fase più operativa, dove attiviamo la generazionedi impulsi, sul pin precedentemente "attaccato" al servomotore, della durata necessaria a posizionare il braccio all'angolo gradi. Si suppone che il servomotore offra una rotazione di 180 gradi e che a 90 corrisponda il punto centrale. Se invece il servomotore può far ruotare liberamente il perno, questo a 90

2011-2016 CC-BY-NC-ND



Sul mercato esistono numerosi modelli di servo motore; le differenze riguardano le dimensioni, la forza esercitata espressa in grammi e la rotazione del perno. Dal sito www.dfrobot.com

gradi sarà immobile e ruoterà a velocità variabile in senso orario da 91 gradi a 180, mentre in senso antiorario da 89 a 0, sempre con velocità progressivamente più elevate. I modelli più diffusi hanno 180 gradi di escursione, ma ci sono modelli da 90, 270 e anche 360 gradi che possono essere reperiti in commercio per applicazioni specifiche; in questi casi la rotazione non corrisponde più al valore fornito in gradi, ma viene espansa o compressa proporzionalmente.

Per passare dai gradi di rotazione a valori assoluti di posizionamento si può usare la funzione **servo.writeMicroseconds(uS)** dove inviamo al servomotore degli impulsi la cui durata è definita esattamente in microsecondi e può quindi riflettere le specifiche del produttore. Solitamente i valori sono fra 1000 e 2000 microsecondi per coprire l'intera gamma di rotazione, ma per tolleranze elettroniche e la mancanza di un effettivo standard questi valori possono estendersi da 700 a 2300 microsecondi. Chi ama la precisione può sperimentare con il proprio servomotore scoprendo a quali valori questo raggiunga il fine corsa da un lato e dall'altro della rotazione. Va infine ricordato che cercare di spostare il motore oltre il fine corsa, continuando ad inviare gli impulsi di durata superiore o inferiore ai suoi limiti, comporta uno "sforzo" che può danneggiare il motore in quanto gli fa assorbire molta corrente.

La funzione successiva **servo.read()** restituisce il valore in gradi utilizzato con servo.write(), semplificando di molto la gestione delle variabili. Non si tratta di una lettura vera e propria della posizione dal servomotore (caratteristica non disponibile con l'hardware dei servo per la modellistica), ma di un accesso ad una variabile gestita dalla libreria per ciascun servomotore definito con il tipo variabile Servo.

Simile comportamento ha **servo.attached()** che restituisce lo stato logico TRUE (o "1") se l'oggetto servo è correntemente attivo su un pin digitale grazie all'istruzione **servo.attach(pin,min,max)**, oppure non è ancora stato attivato o, come vedremo nel prossimo paragrafo, è stato scollegato dal pin.

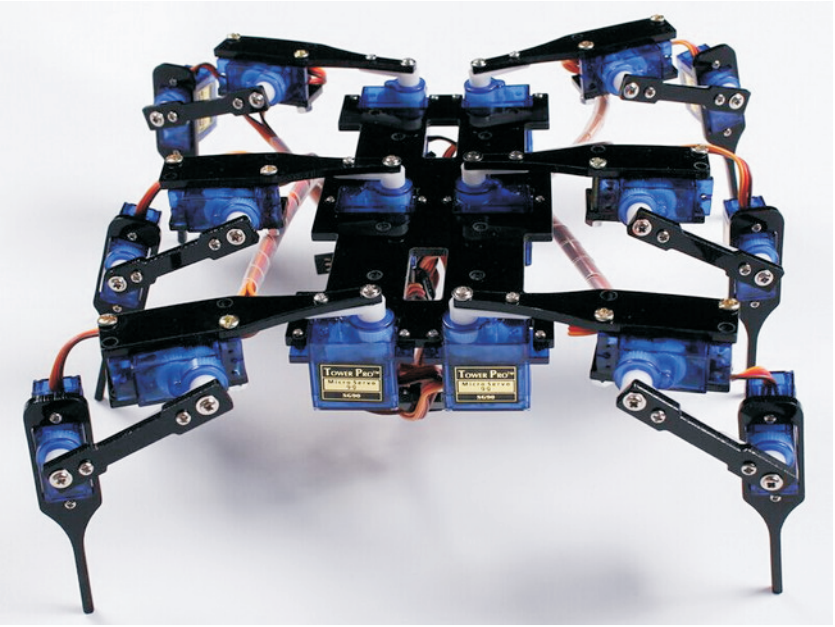
L'ultima funzione è **servo.detach()** e permette di scollegare virtualmente dal proprio pin un servomotore. Dato che è improbabile che un utente scolleghi e ricolleghi i servomotori mentre il software è in esecuzione, la ragione per cui è stata prevista questa funzione è un'altra. Quando si attiva il primo servomotore con la funzione servo.attach(), viene inibita la funzione PWM sui pin 9 e 10, anche se questi non sono utilizzati con dei servomotori. Si tratta di un vincolo della libreria che ricorre alle medesime funzionalità interne del microcontrollore ATmega per

2011-2016 CC-BY-NC-ND

avere impulsi di durata precisa e definita. Guardando infatti il meccanismo di controllo della posizione tramite la durata di un impulso ripetuto nel tempo, possiamo dire che si tratta di una variazione sul tema del PWM che, pertanto, sfrutta proprio il PWM del microcontrollore per la generazione.
Finché ci sono servomotori "attached" la funzione analogWrite() non ha alcun effetto, ma può essere ripristinata scollegando tutti i servomotori. Le operazioni di attach e detach possono essere fatte tutte le volte che si ritiene necessario, inibendo o ripristinando il PWM sui pin 9 e 10.

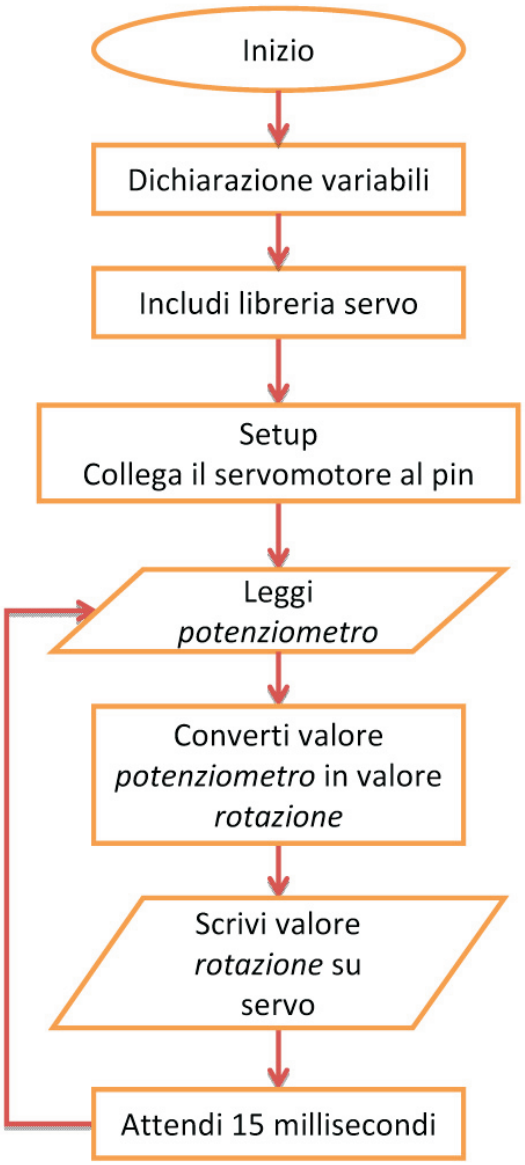
Con i servo ci vuole pazienza

Nella pratica, quando si vuole spostare un servomotore in una posizione si deve tenere conto del fatto che non si tratta di un processo istantaneo e che il servomotore non ha una memoria per la durata dell'impulso ricevuto. Con la funzione servo.write() attiviamo la generazione dell'impulso di durata specificata per un tempo indefinito, ovvero finché non la chiamiamo nuovamente con un nuovo valore o non scolleghiamo virtualmente il servomotore dal suo pin. Il comportamento è simile ad analogWrite(), con la differenza che con quest'ultimo il valore "0" spegne il segnale PWM mandandolo a 0V in modo costante, mentre con il servomotore impostando il valore "0" si va alla posizione iniziale della rotazione e si continua a mantenere la generazione dell'impulso sul pin associato. Finché l'impulso continua ad essere generato sul pin, il servomotore ruota fino al raggiungimento della posizione e la mantiene, ma se l'impulso varia in durata, è necessario tener conto della velocità con cui il motore riesce a spostare il proprio asse. Ogni servomotore ha fra le proprie caratteristiche la velocità di rotazione - espressa solitamente in secondi per grado di rotazione - e questo permette di calcolare il tempo per passare da una posizione all'altra. Pochi gradi corrispondono a pochi centesimi di secondo, mentre rotazioni più ampie richiedono più tempo.
Se scrivi delle applicazioni con i servomotori sarà quindi una buona idea proporzionare il ritardo fra un cambio di posizione e l'altro in base all'ampiezza del movimento e in questo la funzione



Con 18 servo motori, questo robot aracnide a sei zampe richiede una programmazione sofisticata per riuscire a muoversi correttamente.

2011-2016 CC-BY-NC-ND



servo.read() diventa molto utile per calcolare quanti gradi di spostamento ci sono fra la vecchia e la nuova posizione.

Il diagramma di flusso

Fra le dichiarazioni iniziali abbiamo come novità l'inclusione della libreria Servo nel nostro codice, poi il flusso logico prosegue con grande linearità: leggiamo il valore del potenziometro, trasformiamo il suo valore da 0 a 1023 in uno fra 0 e 180, quindi comandiamo il servomotore e attendiamo un po' perché raggiunga la posizione.

La funzione per rimappare i valori

Il fatto che l'ingresso analogico funzioni a 10 bit e le uscite PWM o i gradi dei servomotori siano su altre scale non è certamente passato inosservato agli sviluppatori di Genuino che, per renderci la vita più semplice, hanno predisposto una funzione matematica apposita. La sintassi è molto semplice:

map(variabile, daMin, daMAX, aMin, aMAX)

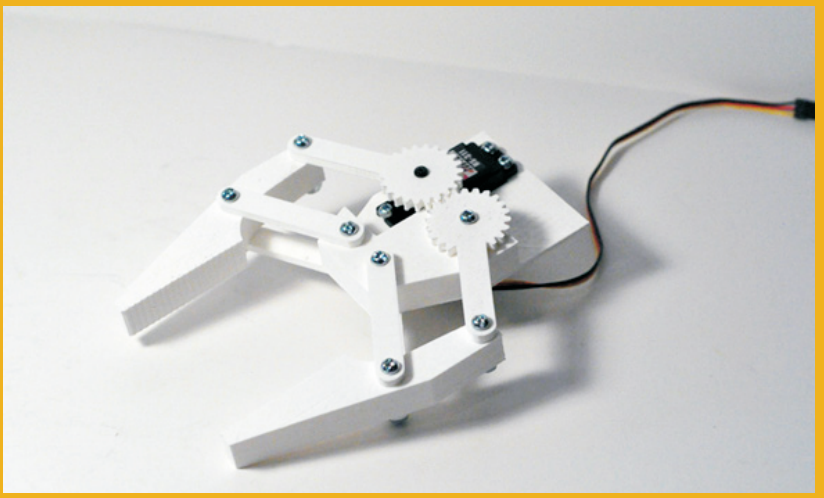
il valore della variabile viene convertito dal suo intervallo di partenza, i cui valori minimo e massimo sono definiti con daMin e daMAX, a quello di destinazione definito con aMin e

```
1  #include <Servo.h>
2
3  Servo Motore1;  // abbiamo creato l'oggetto Motore1
4
5  int Pot = 0;    // il pin di collegamento del potenziometro
6  int Posizione;  // la posizione del cursore del potenziometro
7  int Gradi;      // la posizione del servomotore
8
9  void setup() {
10   Motore1.attach(9); // Il servo è collegato a D9
11 }
12
13 void loop() {
14   Posizione = analogRead(Pot); // acquisiamo la posizione del cursore
15   Gradi = map (Posizione, 0 , 1023, 0, 179);  // ecco il mapping sui gradi
16   Motore1.write(Gradi);       // spostiamo il motore nella nuova posizione
17   delay(15);                  // attendiamo un po' prima di ricominciare
18 }
```


2011-2016 CC-BY-NC-ND

Diamo una mano al servomotore

Le stampanti 3D permettono di realizzare oggetti partendo da modelli tridimensionali e anche il settore della robotica ha tratto grande beneficio da questa tecnologia. Qui vedete una pinza da abbinare a un servomotore. Trovate un modello adatto al nostro servomotore SG-09 sul sito **Thingiverse** che è la repository di modelli 3D più nota del web. Il file vi permette di realizzare una pinza simile a questa.



aMAX ai due estremi. Si tratta di calcoli effettuati in matematica intera e quindi valgono le regole di arrotondamento per troncatura dei decimali.
Sotto il profilo della matematica, lasciamo a voi il divertimento di riprodurre la formula, ricordando come alle scuole medie le proporzioni e i rapporti vi permettevano di passare da una scala all'altra.

Il nostro sketch

Per poter utilizzare la libreria Servo, dobbiamo mettere all'inizio dello sketch la direttiva **#include <Servo.h>**, oppure utilizziamo la funzione dell'IDE che permette di ottenere il medesimo risultato con pochi clic del mouse scegliendo **Sketch -> Inclusione Librerie... -> Servo**. Il resto del codice potete ricopiarlo dal box della pagina precedente.

Il collaudo

Se avete effettuato nel modo corretto i collegamenti dell'hardware, premendo il tasto di upload trasferirete il codice sulla scheda e dopo qualche istante il motore dovrebbe spostarsi in qualche posizione, corrispondente a quella del potenziometro. Ruotando il potenziometro, il perno seguirà il movimento fino a raggiungere la nuova posizione. Se ruotate il perno più velocemente del motore, questo cercherà di seguire lo spostamento. Se avete instabilità nella posizione letta per il potenziometro, potete aumentare il delay() di qualche unità. Questa soluzione non vale se è proprio il potenziometro ad essere "rumoroso" a causa di polvere sporcizia.

Come andare avanti?

Potete proseguire sulla strada della robotica aggiungendo altri servomotori, altri potenziometri, dei sensori e magari anche dei motori tradizionali. Avete imparato a gestirli e a scrivere i vostri primi programmi, quindi avete a disposizione le basi per sviluppare dei progetti seguendo la vostra creatività.
Siete anche giunti al termine di questo testo e speriamo di avervi guidato sin qui riuscendo a trasferirvi le basi metodologiche della programmazione e della prototipazione. Ora vi aspettano altre ore di divertimento utilizzando i numerosi esempi che sono già presenti nel software che avete installato; ricordate che tutte le spiegazioni di questi esempi, inclusi i relativi circuiti da utilizzare per farli funzionare, sono sul sito Arduino.cc (**https://www.arduino.cc/en/Tutorial/HomePage**).

2011-2016 CC-BY-NC-ND

Il testo è terminato e spero vi sia stato utile per iniziare ad esplorare le possibilità offerte dalla piattaforma elettronica open-source Arduino e le sue schede Genuino. Se volete dimostrarmi che avete gradito il mio lavoro di divulgazione, potete farlo cliccando sull'icona qui sotto e inviando quello che vorrete. Nel caso decidiate di farlo, vi ringrazio anticipatamente.

Simone Majocchi

PayPal™
Clicca qui per una donazione

Except where otherwise noted, this work is licensed under
http://creativecommons.org/licenses/by-nc-nd/3.0/